David Fernández Marquez

IMPLEMENTACIÓN DE UN SISTEMA DE ACTUALIZACIÓN AUTOMÁTICA DE ANUNCIOS EN GOOGLE ADS

Trabajo de fin de Grado

Dirigido por Marc Sánchez Artigas

Grado en Ingeniería Informática

UNIVERSITAT ROVIRA I VIRGILI

Tarragona
2021

Resumen

En este proyecto se ha creado un sistema independiente que permite la creación y mantenimiento de una colección de anuncios en Google Ads para el portal de ofertas limitadas Buscounchollo.com. Este sistema se ha implementado en un docker individual en Amazon Elastic Container Service a partir de un proyecto Symfony completamente independiente, facilitando su actualización, escalado, y mantenimiento.

El sistema permite la creación y mantenimiento de un número arbitrario de anuncios, agrupados en AdGroups y definidos mediante simples schemas json. El sistema permite la sincronización periódica de forma automática de los anuncios, analizando que anuncios necesitan ser creados, actualizados, o cerrados, manteniendo paridad entre la base de datos de la empresa y los anuncios en los servidores de Google sin requerir intervención de seres humanos en ningún momento del proceso.

Abstract

This project presents an independent system to allow creation and upating of a collection of Google Ads ads for the travel bargain web portal Buscounchollo.com. The system has been built as an isolated Symfony project and implemented in an individual docker hosted in Amazon Elastic Container Services, thus facilitating later updates, upscaling, and maintenance.

The constructed system allows for the maintenance of an arbitrary number of ads, grouped by AdGroups and built with text patterns defined by simple json schema. It allows for constant periodic synchronization between the created ads and the company systems, analyzing whether ads need to be newly created, updated, or paused/removed. This lets us keep complete data parity between the company systems and the published ads without requiring human intervention at any point in the process.

Resum

En aquest projecte s'ha creat un sistema independent que permet la creació i manteniment d'una col·lecció d'anuncis en Google Ads per al portal d'ofertes limitades Buscounchollo.com. Aquest sistema s'ha implementat en un docker individual en Amazon Elastic Contenidor Service a partir d'un projecte Symfony completament independent, facilitant la seva actualització, escalat, i manteniment.

El sistema permet la creació i manteniment d'un nombre arbitrari d'anuncis, agrupats en AdGroups i definits mitjançant simples schemas json. El sistema permet la sincronització periòdica de manera automàtica dels anuncis, analitzant quins anuncis necessiten ser creats, actualitzats, o tancats, mantenint paritat entre la base de dades de l'empresa i els anuncis en els servidors de Google sense requerir intervenció d'éssers humans en cap moment del procés.

Contenido

Índice de figuras

1. Introducción

En este trabajo, mi intención es diseñar y desarrollar un sistema automático que permita mantener una campaña de Google Ads para el sitio de ofertas Buscounchollo.com, de forma automática y sin requerir acciones por usuarios excepto en casos de políticas incompatibles.

No es ningún secreto que en internet, a día de hoy, el negocio de la publicidad es un duopolio – si una compañía pretende anunciarse en internet, las únicas opciones reales son Google y Facebook. Otras opciones menores *existen*, pero la diferencia en marketshare es tan desmesurada que pueden considerarse insignificantes. Y ambas compañías se encuentran enzarzadas en una competencia constante para atraer más anunciantes.

En este intento de atraer más anunciantes, Google está refinando constantemente sus sistemas de anuncios, dando a los anunciantes más opciones, más datos con los que tomar decisiones, más recomendaciones, refinando cada vez más el targeting… así como una interfaz gráfica extremadamente completa.

Por un lado, esto permite a un anunciante tener un control extremadamente detallado de sus anuncios, sus pujas, sus budgets, políticas de anuncios, y otros factores que son las armas en la constante batalla de los anunciantes por atraer clicks.

Por otro lado, este mismo nivel de control tan detallado hace que si una empresa pretende tener una gran cantidad de anuncios, esto requiera una enorme cantidad de trabajo, creando y revisando anuncios en la interfaz web de Google, revisando métricas, reescribiendo anuncios conforme las circunstancias cambian para maximizar clicks, etcétera.

Por ello, la perspectiva de poder automatizar al menos las tareas más costosas (especialmente la creación y revisión de anuncios) es extremadamente atractiva. Sin embargo, no es un desarrollo trivial, ya que la interfaz de Google es compleja y restrictiva. Así que la mayoría de empresas se conforman con hacer los anuncios a mano, y perder esa gran cantidad de horas. De acuerdo con los responsables de marketing de Viajes Para Ti, estiman que no más de 50 empresas españolas tengan una integración completa con Google a nivel de anuncios.

Dado todo esto, cuando se me presentó la oportunidad de crear una integración con Google Ads, no pude sino aprovecharla.

2. Sobre la empresa

Para entender el proyecto y por qué se han tomado las decisiones que se han tomado, dedicaré un momento a explicar la empresa para la cual se ha creado, ya que es difícil entender las necesidades de un cliente si no se entiende la naturaleza de éste.

La empresa en la que se ha desarrollado este proyecto es Viajes Para Ti SL, una empresa del sector turístico cuyo principal negocio es proporcionar diferentes tipos de viajes a diferentes segmentos de mercado a través de sus tres portales web.

A día de hoy, la empresa se ubica en la posición 44 del ranking de Ecommerce España

La empresa tiene tres marcas comerciales contenidas bajo su bandera:

Ilustración 1: Marcas Viajes Para Ti

Esquiades.com es el portal original – la compañía nació como Esquiades SL, de hecho. Es un portal dedicado principalmente a la venta de alojamientos y forfaits, así como alquiler de equipo y otros complementos necesarios para el deporte de montaña, para temporadas de nieve. El portal está integrado con la mayor parte de los grandes proveedores de nieve en España y Andorra, pero está en una fase de expansión en la cual pretende ofrecer producto de montañismo y esquí en toda Europa.

Buscounchollo.com es, actualmente, el más visitado de los portales de la empresa. A diferencia de Esquiades, donde el usuario puede decidir los detalles de su viaje, Buscounchollo.com es un sitio de ofertas (que se denominan "chollos") predefinidas, con fechas limitadas.

Estas fechas se acuerdan con proveedores de antemano y se empaquetan, ya que la idea del sitio es, sencillamente, ofrecer **el mejor precio que existe en internet para una combinación concreta**. Es decir, si en Buscounchollo tenemos una oferta de 3 días en el hotel X con régimen Y en un cierto rango de fechas, es porque el hotelero se ha comprometido a que no está proporcionando mejores precios para el hotel X en régimen Y durante estas fechas a ningún otro portal o agencia de viajes. Lógicamente, la cantidad de chollos disponibles fluctúa durante el año, pero el portal raramente tiene menos de 100 chollos diferentes disponibles.

Buscounchollo está casi completamente dedicado al público nacional.

Amimir.com es el nuevo portal de la empresa, que nace principalmente con la intención de aprovechar todos los contactos que la empresa ha ido desarrollando en los últimos años

con entidades hoteleras. Es un buscador de hoteles, que permite al usuario buscar qué hoteles tienen disponibilidad en ciertas fechas, y encontrar el precio más barato disponible entre todas las opciones. El propio portal ya realiza a cabo comparativas entre todo el producto ofrecido por hoteles en redes de integración, y en caso de que el usuario decida reservar, Viajes Para Ti se encarga de hacer todas las gestiones con el hotel, haciendo de intermediario.

En este proyecto, mi principal interés es Buscounchollo.com, ya que es éste el portal para el cual implementaré el sistema de anuncios.

Un detalle importante, a la hora de afrontar este trabajo, es que la empresa ya utiliza Google Ads muy fuertemente, si bien de forma manual – el equipo de Marketing de la empresa dedica una parte muy importante de su jornada a crear y mantener anuncios. Actualmente, la media mensual de inversión de la empresa en los sistemas de publicidad de Google (Ads de búsqueda, anuncios YouTube, anuncios institucionales) ronda los 100.000€, llegando en los meses de temporada máxima a rozarse los 200.000€. Esto es un beneficio importante, a la hora de plantear el trabajo, ya que permitirá que la empresa tenga muy claras sus necesidades con respecto a este proyecto.

3. Sobre el sistema Google Ads

Para entender este proyecto, es importante entender qué es Google Ads, y al menos la estructura básica de trabajo, a nivel de usuario. De esta forma, cuando hable sobre las entidades de la API en secciones posteriores, éstas tendrán más sentido.

3.1 Qué es Google Ads

Citando la propia definición de Google, "Google Ads es el programa de publicidad en línea de Google. A través de Google Ads, puede crear anuncios en línea para llegar a las personas en el momento exacto en que se interesan por los productos y servicios que ofrece"[1]. Es el sucesor al sistema de anuncios Google AdWords.

En términos menos publicitarios, Google Ads es la nueva interfaz que Google pone a disposición de las empresas para crear anuncios de pago que aparezcan en las búsquedas realizadas por usuarios. El lector habrá visto sin duda los anuncios resultantes de Ads y AdWords, que tienen este aspecto:

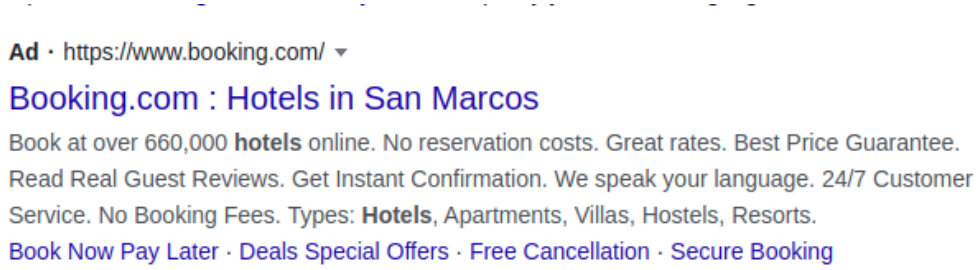

Ilustración 2: Anuncio en buscador

Nótese la cabecera indicando "Ad", que marca el resultado como un anuncio de pago y lo diferencia de los resultados orgánicos de búsqueda (se considera un resultado orgánico de búsqueda aquel que se muestra puramente por decisión del motor de búsqueda, sin prioridad dada a clientes que pagan).

El mecanismo de funcionamiento, a nivel teórico, de Google Ads es más o menos como sigue:

- El usuario A (anunciante) crea un anuncio en Google Ads, y le asigna unas palabras clave o términos de búsqueda, así como un CPC (coste por click) que está dispuesto a pagar cuando los usuarios hagan click en el anuncio.

- El usuario B (persona que busca) realiza una búsqueda en la cual aparece una o más de las palabras clave elegidas por A.

- El anuncio se muestra a B, de la forma vista en la ilustración anterior.

- Si B hace click en el anuncio (es decir, el anuncio ha conseguido un clickthrough), A paga a Google el CPC acordado.

---

1 Ver https://support.google.com/google-ads/answer/6319?hl=es-419

En principio, sencillo, y muy efectivo a nivel de coste para A, ya que **A solo paga a Google si el anuncio ha tenido éxito** y ha conseguido que el usuario entre a su portal web.

Sin embargo, en una página de búsqueda solo hay, por lo general, espacio para **dos anuncios de pago**, tres a lo sumo, mientras que puede haber cientos de anuncios de Google Ads intentando apuntar a las mismas palabras clave. Por esto, en realidad Google Ads funciona mediante un **sistema de pujas** para decidir qué anunciante consigue el espacio.

Cuando A crea el anuncio, no decide un CPC fijo, sino que lo que indica a Google es **el CPC máximo que A está dispuesto a pagar por un click**. Cuando un usuario hace una búsqueda que toca ciertas palabras clave, el sistema lleva a cabo una subasta entre todos los anuncios que tengan estas palabras clave, y el anunciante que esté dispuesto a pagar más por el click consigue que se muestre su anuncio.

Por ejemplo, si tenemos 3 anunciantes A, B y C, cuyos CPCs máximos para una cierta palabra clave son respectivamente 0,32, 0,25, y 0,28, el anuncio que se mostrará será el de A, y si el usuario buscador hace click, A pagará 0,29 (para superar la puja de C).

Por supuesto, esto es a nivel teórico. A nivel práctico, esto está modulado por muchos factores adicionales. Por ejemplo, el sistema de decisión de Google tiene en cuenta no solo el CPC dispuesto a pagar, sino también el ratio muestras/clicks. Si el anuncio de A, por estar mal escrito o no ser atractivo, se muestra muchas veces pero no recibe una cantidad proporcional de clicks, Google empezará a darle prioridad al siguiente anuncio en las pujas, en este caso el de C.

3.2 Estructura conceptual de Google Ads

Google Ads, al ser vender anuncios una de las principales ocupaciones de Google, es un sistema extremadamente complejo, pero por suerte, para el proyecto que nos ocupa solo nos interesa una pequeña subsección, de la cual voy a intentar dar un breve esquema en las siguientes páginas.

Google Ads guarda sus entidades en un sistema jerárquico, bastante similar al de la estructura de un sitio web. Un elemento en Google Ads se identifica mediante lo que se llama un **resource name** – básicamente, una URI que identifica de forma inequívoca el recurso, así como los niveles superiores de la jerarquía de los cuales depende.

Un resource name puede tener, por ejemplo, la siguiente forma:

customers/4783828488/campaigns/121144866916

Esto indica que esta es la campaña con id 121144866916, que depende del Customer (usuario) 4783828488. Lógicamente, un resource name, al ser prácticamente una ruta específica, es completamente único para cada elemento.

A nivel de programación, es importante tener en cuenta que **a menudo hay una diferencia entre la jerarquía real de un objeto, representada en su resource name, y la forma en que se organizan las cosas en la metáfora** de la interfaz gráfica de Ads,

que es como se entienden por el usuario. En este apartado hablaré solo del nivel conceptual.

(Todas las imágenes en este apartado salen de la campaña creada con este mismo sistema, para mostrar el resultado real)

3.2.1 Customers

Un mismo usuario de Google puede tener múltiples "usuarios" de Ads, por ejemplo para poder mantener separadas sus marcas. Estos pseudo-usuarios se llaman Customers, y cada uno tiene una credencial de Ads propia.

Esto permite, por ejemplo, que el administrador de una cuenta de empresa pueda permitir a diferentes miembros de la empresa acceso a las entidades de diferentes Customers, sin necesidad de permitir acceso completo a los demás.

Customer es el elemento "raíz" de la mayoría de los elementos de Ads.

3.2.2 Campañas

Cada customer puede tener un número cualquiera de campañas. La metáfora de la campaña de Ads es básicamente lo que puede suponerse – una campaña publicitaria, con una cierta estrategia y un objetivo.

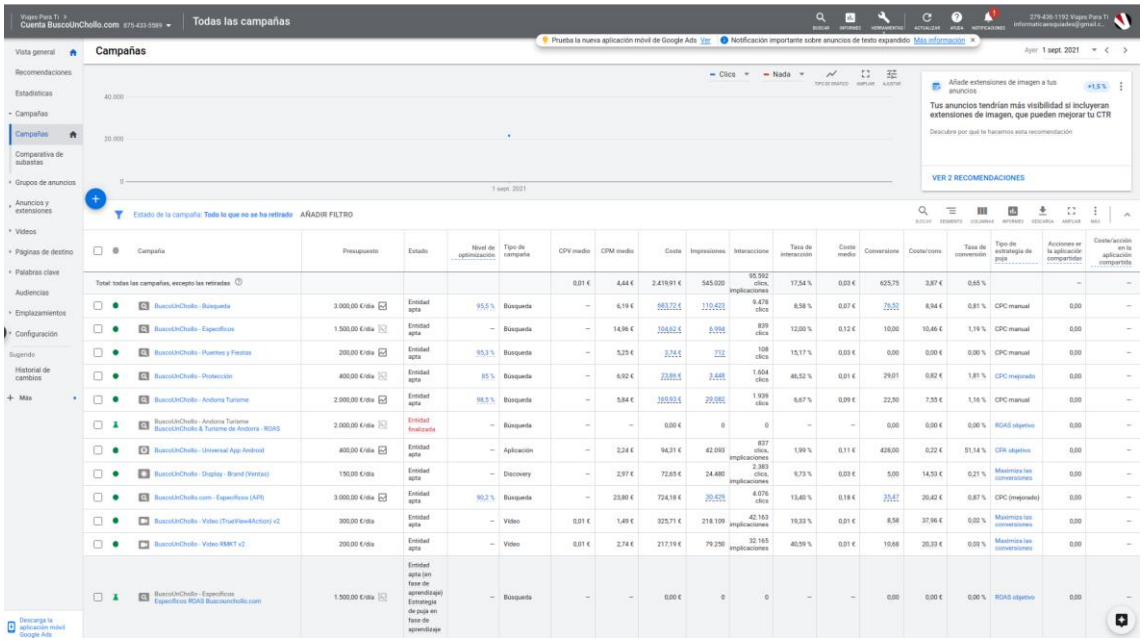

Ilustración 3: Google Ads GUI (campañas)

Una campaña puede ser de múltiples tipos [2]: campañas de búsqueda, de vídeo, contextuales… A nivel de este proyecto, ya que queremos hacer anuncios de texto que aparezcan en búsquedas estándares, estamos trabajando siempre con campañas de búsqueda.

2 Ver https://support.google.com/google-ads/answer/2567043 para una descripción de los tipos de campaña y sus estrategias.

3.2.3 Grupos de anuncios

Entrando dentro de una campaña, tenemos los AdGroups, grupos de anuncios. A nivel de metáfora, los usuarios de marketing con los que he podido hablar tienden a considerar los AdGroups como los "anuncios" (así que cuando se habla de "crear un anuncio", generalmente lo que se quiere decir es "crear un AdGroup con todas sus entidades relacionadas"), pero a nivel de organización, lo que realmente hace un AdGroup es hacer de unión entre una serie de anuncios, palabras clave, y políticas de puja.

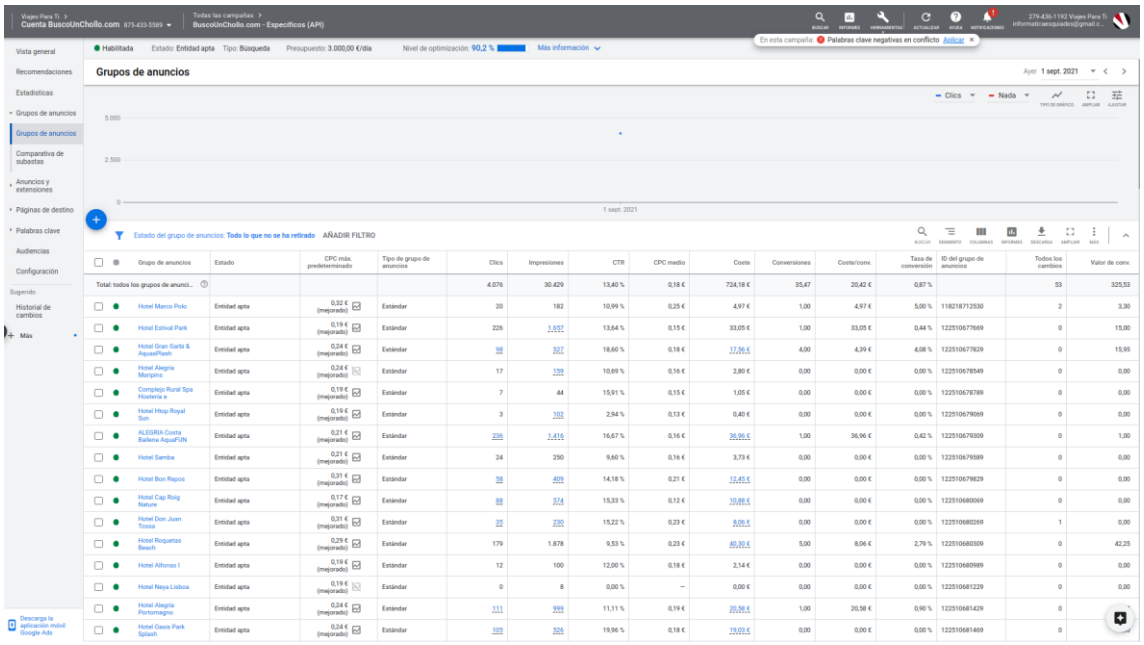

Ilustración 4: Google Ads GUI (AdGroups)

3.2.4 Anuncios en un grupo de anuncios

Entrando al AdGroup, lo primero que encontramos son los Ads – las entidades que realmente representan los anuncios que se mostraran al usuario. Un Ad puede tener políticas y CPC específicos, que sobrescriben los que se hayan designado en el AdGroup.

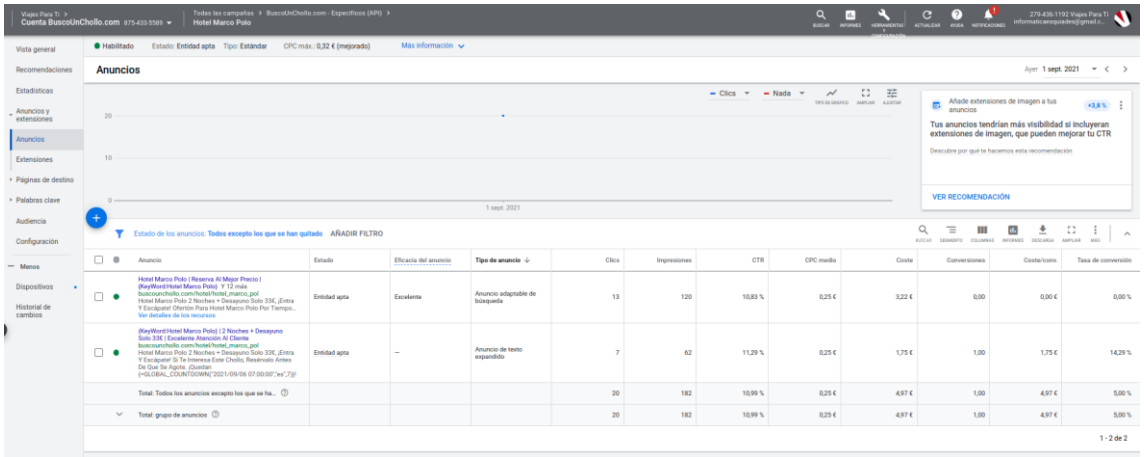

Ilustración 5: Google Ads GUI (Ads)

Veamos una pantalla de un Ad (en este caso, un Ad de tipo Adaptable):

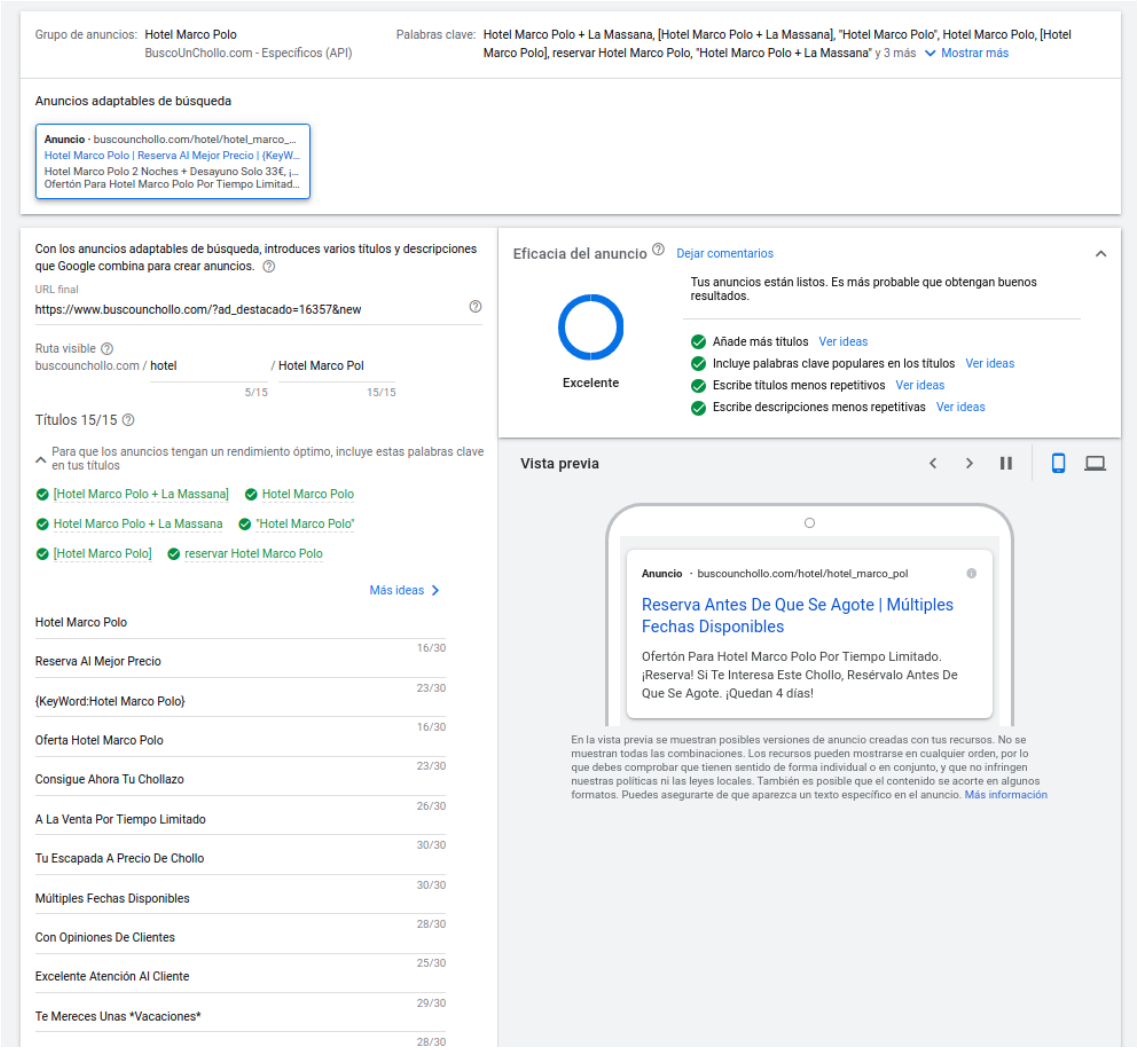

Ilustración 6: Google Ads GUI (Pantalla de anuncio)

Podemos ver que un anuncio incluye una url a la que apunta el anuncio y a donde irá el usuario al hacer click, un path visible (si se recuerda la ilustración 2, es el texto que aparece encima del anuncio, al lado del marcador de Ad), una serie de titulares, y una serie de descripciones. En la esquina superior derecha se muestran las keywords del grupo que se aplican para mostrar este anuncio, y en la inferior derecha una vista previa del aspecto del anuncio para poder comprobar como se verá realmente en el buscador.

3.2.5 Palabras clave

Como se ha comentado, las palabras clave se crean a nivel de AdGroup, pese a ser los elementos que deciden si se muestra un Ad.

Haciendo click en un AdGroup, podemos ver todas las palabras clave actualmente introducidas para este AdGroup:

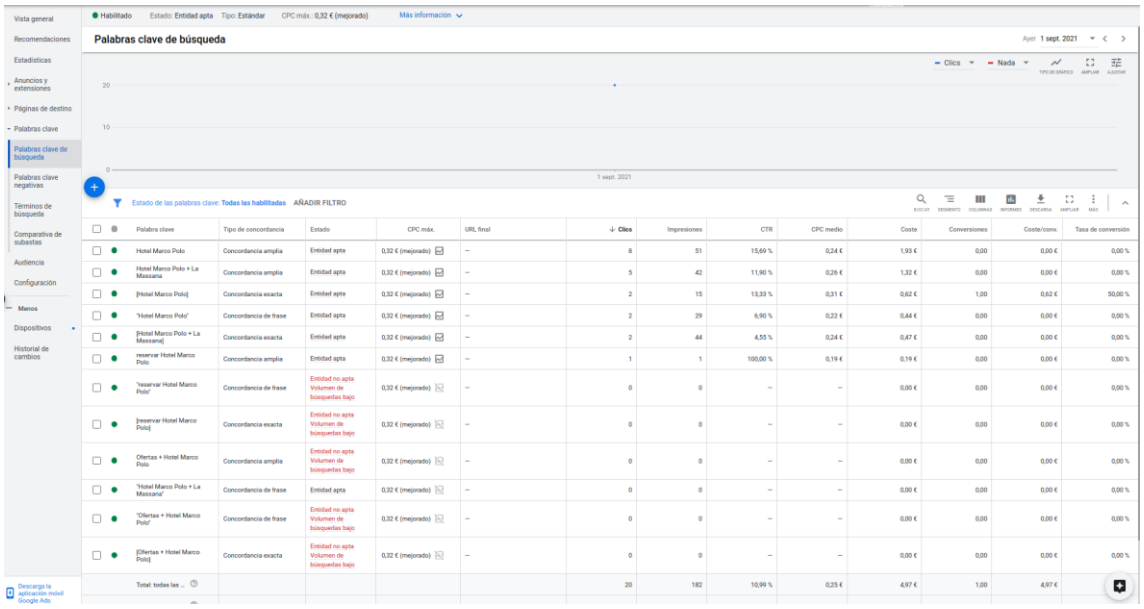

Ilustración 7: Google Ads GUI (Palabras clave)

A nivel de metáfora, las palabras clave son el decisor principal de si un anuncio se muestra o no. A nivel de entidad las palabras clave son simplemente un tipo de Criterion (Google proporciona muchos tipos de Criterion) pero en la metáfora visual los separa en pantallas diferentes.

Una palabra clave es básicamente una expresión regular con una serie de reglas estandarizadas que se aplican dependiendo del tipo de concordancia: amplia, de frase, o exacta.

4. Situación inicial y objetivos

Antes de entrar en detalles sobre el diseño específico, quiero establecer de forma clara las necesidades que habrá de suplir, y cuáles son los objetivos tanto prácticos como formativos del proyecto.

4.1 Situación:

Como se ha comentado en la introducción, el portal Buscounchollo.com ofrece ofertas prediseñadas. En un momento dado, el portal suele mostrar entre 100 y 150 ofertas ("chollos"). Estas ofertas rotan constantemente, conforme los proveedores ofrecen nuevas oportunidades a la empresa y las ofertas se agotan o cambian sus condiciones.

Como puede deducirse inmediatamente, esto hace que mantener una selección completa de anuncios de Google Ads actualizados manualmente para todos los chollos, de la misma forma que se hace para el portal de Esquiades.com, sea impracticable.

Sin embargo, la simple realidad es que, de acuerdo a Nacho Vallina, responsable de Marketing de Viajes Para Ti con quince años de experiencia tratando con venta online, para portales de viajes la media del tráfico que entra a través de anuncios de Google supera el **30%** de los clicks (ver ilustración a la derecha para un gráfico creado con datos de VPT durante 2019 – y, por contexto, quiero recordar al lector que esto es con un sistema de Google Ads manual). Lógicamente, no tener presencia en Google Ads para un portal entero, dadas estas cifras, era impensable.

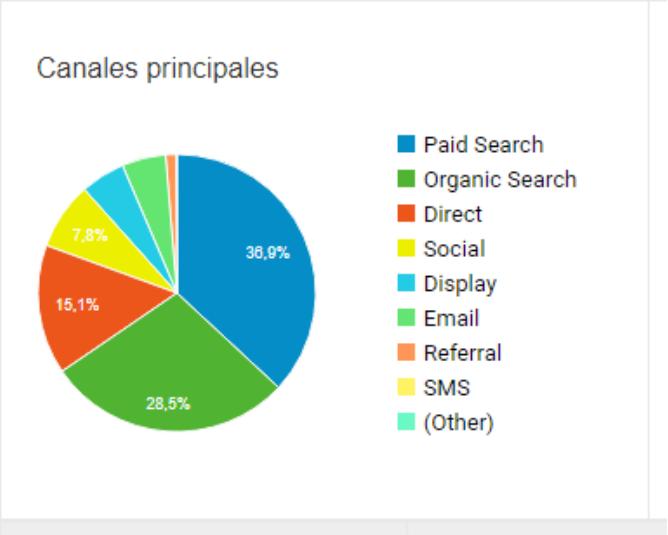

Ilustración 8:Porcentajes de procedencia de trafico de Enero a Junio (2019) en Esquiades.com

Como un punto medio, lo que la empresa hacía hasta ahora para Buscounchollo.com era una doble estrategia. Por un lado, mantener una enorme cantidad de anuncios genéricos raramente modificados que atrajeran posibles clientes sin targeting muy afinado. Por otro lado, hacer que el equipo de marketing seleccionase alrededor de 50 chollos que considerase máximamente marketables, creara anuncios para estos chollos, y fuese responsable de mantener los anuncios actualizados. Ni que decir tiene, esta segunda estrategia es efectiva pero tremendamente ineficiente, tanto a nivel de tiempo (según entrevistas con el equipo de marketing, una semana media podía implicar **una media de entre 8 y 10 horas** dedicadas únicamente a revisar anuncios de forma manual, con un incremento de más del 50% en temporada alta) como a nivel de coste (entre que un chollo salga de ventas y un ser humano se diese cuenta y editase el anuncio, podían pasar horas o incluso, en fines de semana, días – y como hemos visto en el apartado anterior, Google

cobra por clic, así que tener visible y recibiendo clicks un anuncio que apunta a un chollo agotado es **coste perdido para la empresa**).

Adicionalmente, otro punto problemático es la tendencia de los equipos que introducen datos en el sistema a **reutilizar los chollos existentes** cambiando los enlaces cuando se obtienen nuevas ofertas para el mismo hotel, dejando los anuncios apuntando a urls que ya no existen. Esto complicaba el proceso de saber manualmente si un chollo había cambiado y necesitaba modificar el anuncio, añadiendo todavía más carga de trabajo.

Por otro lado, la empresa ya había hecho algún intento de hacer algunos elementos automáticos – para el portal de Amimir.com para empezar. Al ser Amimir.com el portal con menor carga, ya que es el portal más nuevo y con menos visitas, se consideró que era un buen campo de pruebas para empezar, y se había construido un sistema sencillo basado en la API de Google AdWords (el antecesor de Google Ads) que permitiera a Marketing escribir anuncios manualmente desde el sistema de la empresa y copiarlos a múltiples hoteles cambiado algunos comodines. Nunca dio un excesivo fruto, ya que requería una cantidad similar de trabajo manual a simplemente hacer los anuncios en la interfaz de Google, pero sería importante crear el diseño teniendo en cuenta que estas funcionalidades podrían querer actualizarse a la nueva API de Google Ads, así que sería importante mantener una cierta similitud para minimizar la dificultad de implementar la retrocompatibilidad de las vistas ya creadas en la medida de lo posible.

4.2 Objetivos del proyecto:

Tras analizar esta situación, los objetivos que se plantearon inicialmente fueron los siguientes:

- **Analizar el sistema de creación de anuncios ya existente** para Amimir.com, y estudiar la posibilidad de aprovecharlo para el nuevo sistema.
- Preparar un **sistema fácilmente extensible** que permitiera crear las entidades básicas de Google Ads (AdGroup, AdGroupAd, Ad, Criterion) mediante código a programadores futuros en la empresa sin necesidad de interactuar directamente con la API de Google, de forma que permitiera en un **futuro crear anuncios para el resto de los portales** de Viajes Para Ti SL.
- **Actualizar automáticamente anuncios existentes** cuyas propiedades hayan cambiado (principalmente urls asociadas) conforme los chollos se actualizan, con el objetivo de eliminar el tiempo que Marketing ha de dedicar a la revisión manual de estos anuncios.
- Crear un sistema que permita la **creación de nuevos anuncios a partir de plantillas prediseñadas** para chollos recién creados, para evitar la necesidad de crear anuncios manualmente.
- Automatizar el **cierre de los anuncios** cuyos plazos se hayan terminado, para eliminar la necesidad de que marketing tenga que comprobar exhaustivamente el estado de los chollos y sus anuncios relacionados.
- **Establecer un sistema de reglas extensible para el pausado** automático de anuncios basado en lógica de negocio.

4.3 Objetivos formativos

A nivel personal, lógicamente, yo entro en este proyecto con la intención de aprender, así que he planteado unos objetivos formativos generales antes de comenzar:

- Google Ads: El objetivo primario. Cuando termine este proyecto, me gustaría poder tener un dominio importante de como funciona no solo la API de Google Ads, sino en general cómo funciona el sistema de anuncios: qué es un anuncio en el sistema, cómo se deciden qué anuncios son “buenos” y cuales son “malos”, como decide Google como mostrar anuncios…
- Google Cloud en general: Lo más probable es que trabajar con la API de Google Ads vaya a requerir entrar en los sistemas de Services y similares, los cuales conozco vagamente pero nunca he tenido oportunidad de utilizar. Esta será una buena oportunidad de hacerlo.
- Sistemas REST y peticiones: durante la carrera solo he tenido la oportunidad de tocar sistemas REST en una única asignatura, y a decir verdad no me siento muy cómodo haciendo curls. Montar un sistema complejo que tendrá que interactuar con la API REST del mayor anunciante del mundo ciertamente parece una buena oportunidad para suplir esta carencia.
- Sistemas de decisiones: El planteamiento inicial del proyecto incluye alguna forma de tomar decisiones sobre cuánto queremos pujar o qué anuncios queremos pausar. Si tengo oportunidad, me gustaría intentar llegar a implementar algo muy básico de inteligencia artificial.

5. Descripción general del proyecto

5.1 Entorno

5.1.1 Arquitectura de la empresa

Los sistemas de Viajes Para Ti SL sobre los cuales se ha montado el proyecto se encuentran divididos entre aproximadamente 20 proyectos individuales. Cada proyecto tiene sus peculiaridades, pero aquí solo entraré en las generalidades.

- Casi todos los proyectos de la empresa utilizan un framework típico de PHP/Symfony, utilizando Doctrine sobre contenedores MySQL para persistir datos.
- La mayoría de los sistemas más nuevos o de menor envergadura se mantienen en instancias ligeras en Amazon Elastic Container Services (ECS), gestionadas por un load balancer que levanta más o menos instancias dependiendo de la carga actual, y mantienen caché entre instancias mediante bases de datos Redis.
- Los sistemas centrales (Esquiades, Buscounchollo), al ser los más antiguos, contienen una gran cantidad de código legacy que dificulta su traspaso a ECS, y se mantienen en un servidor central.
- Cada proyecto se mantiene en un entorno docker con una imagen Ubuntu específicamente preparada con las versiones de php y extensiones requeridas, configuraciones individuales, archivos de credenciales exclusivos al proyecto, etc.
- Los proyectos no tienen comunicación directa ni constancia entre sí – para transmitir datos entre ellas se utilizan peticiones html POST y GET.
- Debido a los diferentes ritmos de desarrollo en diferentes proyectos, no hay estandarización de versiones entre proyectos.

Originalmente, se había pretendido insertar este proyecto en el servidor central de Esquiades, donde se había ubicado originalmente el intento de integración con Google AdWords. Sin embargo, durante la primera implementación, se descubrió que esto no era factible, debido a las necesidades del proyecto, y por tanto se decidió que se crearía un nuevo proyecto GoogleAdsProject para hospedar este sistema, así como las modificaciones futuras que se hicieran.

5.1.2 Sobre la API de Google Ads

Google ofrece una documentación muy completa[3] de la API de Google Ads, así que no me extenderé en detalles aquí. Sin embargo, para entender el proyecto en general, es importante entender al menos la naturaleza de la API con la que habremos de comunicarnos.

Google Ads API es un endpoint que Google ofrece para poder realizar las mismas operaciones que se ofrecen en la interfaz gráfica de GAds mediante peticiones html. A bajo nivel, es sencillamente una interfaz REST, la cual recibe objetos JSON y devuelve objetos JSON. Técnicamente existe la posibilidad de enviar llamadas curl directamente a este endpoint si se tienen las credenciales, pero la documentación de Google insiste en que **no se utilice el endpoint REST directamente**. En vez de esto, se recomienda

3 Ver https://developers.google.com/google-ads/api/docs/start

encarecidamente que se haga uso de las librerías proporcionadas en la misma página de la documentación para los lenguajes más comunes. La librería de php es **googleads/google-ads-php**.

Comunicarse con este endpoint requiere, por regla general, lo siguiente:

- Un set de credenciales cifradas.
- Una identificación del tipo de elemento o elementos con lo que interactuaremos.
- Una identificación del tipo de acción a llevar a cabo.
- Los resource names de los elementos con los que pretendamos interactuar (o una query GQL que permita seleccionar un rango de elementos para obtener sus resource names).

La ventaja de usar las librerías prediseñadas es que permiten trabajar de forma orientada a objetos, y por tanto los puntos 2 y 3 de esta lista ya están inherentemente identificados por los diferentes tipos de objetos y los métodos utilizados, y el cliente de Google puede reconocerlos y preparar el curl apropiado de forma transparente.

5.1.3 Necesidades y requisitos

Todas las necesidades del proyecto surgieron de una sola necesidad básica: como se ha visto en el apartado anterior, para poder comunicarnos con la API de Google Ads en php, siguiendo los estándares que Google requiere, se necesitan las librerías **googleads/google-ads-php** (**versión 9.0** o superior, al estar las anteriores deprecadas), y la librería **google/apiclient, versión 2.5** o superior.

El problema que se encontró es que ambas librerías, para utilizarse dentro del framework Symfony, requieren un mínimo de **php 7.4** y **Symfony 4.4**, mientras que el docker de Esquiades se encuentra, debido a dependencias legacy, en una situación donde incrementar versión de Symfony por encima de 3.4 y php por encima de 7.1 es impracticable a corto plazo debido a la cantidad de recursos de desarrollo que esto requeriría.

Por tanto, se decidió crear un nuevo docker para alojar el sistema de comunicación con Google Ads. Una vez decidido esto, nuestra lista de necesidades se pudo solidificar a la siguiente lista:

- Un set de credenciales de Ads/AdWords para Buscounchollo (dado que la empresa ya tenía credenciales AdWords para Amimir, la mayoría de credenciales ya estaban disponibles)
- Nueva imagen de Ubuntu, con los siguientes requisitos mínimos preinstalados:
    - Php 8.0
    - Xdebug 3.0
    - PECL gRPC
    - PECL Protobuf
- Nueva base de datos SQL para guardar datos que se persistirán
- Nuevo Docker, con la imagen de Ubuntu montada y acceso a la nueva base de datos
- Librerías Composer **mínimas** requeridas por el proyecto:
    - "doctrine/annotations": "^1.12",

- "doctrine/doctrine-bundle": "^2.3",
- "doctrine/doctrine-migrations-bundle": "^3.1",
- "doctrine/orm": "^2.8",
- "google/apiclient": "^2.9",
- "google/protobuf": "^3.17",
- "googleads/google-ads-php": "^9.0",
- "sensio/framework-extra-bundle": "^6.1",
- "symfony/console": "5.2.*",
- "symfony/dotenv": "5.2.*",
- "symfony/flex": "^1.3.1",
- "symfony/framework-bundle": "5.2.*",
- "symfony/monolog-bundle": "^3.7",
- "symfony/proxy-manager-bridge": "5.2.*",
- "symfony/yaml": "5.2.*"

5.2 Previsiones de uso

A nivel básico, la previsión de uso es simple: se espera que se llame a la función que **actualizará los anuncios y pausará los terminados de forma automática**, cada hora, mediante un script cron en el cual **no intervendrá ningún usuario**.

Sin embargo, como se ha dicho, previsiones de uso a futuro implican la creación de muchos más casos de uso conforme se expanda el scope del proyecto una vez este TFG se haya presentado. Por tanto, se puede esperar, a nivel futuro, que sean necesarios casos de uso independientes para crear anuncios específicos, para obtener anuncios concretos, y en general para **crear un sistema que haga fácil añadir a posteriori diferentes formas de que usuarios puedan interactuar con los anuncios existentes en la plataforma de Ads**[4].

4 De hecho, después de terminar el proyecto explicado en este documento pero antes de terminar esta documentación, ya hemos añadido nuestra primera pantalla de usuario, una pantalla que obtiene todos los actualmente activos mediante este proyecto, y después compara con la base de datos de Buscounchollo para obtener la facturación real - y hay alguna más proyectada.

6. Diseño básico (casos de uso)

Dado que la idea del proyecto no se centra tanto datos específicos como en cumplir un caso de uso concreto (poner al día todos los anuncios de Buscounchollo en una sola instrucción) pero de forma que los componentes individuales se puedan reutilizar, el diseño de la parte principal del sistema no se ha hecho partiendo de un diagrama de clases (ya que no se sabía, en un principio, que clases harían falta), sino que se ha hecho top-down, basado en lo que podríamos llamar "casos de uso concéntricos".

Es decir, en lugar de tener muchas utilidades separadas como casos de uso independientes, **se ha planteado el caso de uso superior final único** (caso de uso D1, en esta documentación) **como objetivo a cumplir** desde el principio, y se han analizado las **acciones individuales que se requerirían para llevar a cabo este caso de uso superio**r como casos de uso, los cuales a su vez requieren otras acciones que pueden considerarse casos de uso individuales que se pueden querer volver a utilizar en un futuro, y así sucesivamente, subdividiendo en acciones **hasta llegar a los elementos "atómicos" del sistema**, casos de uso que no tiene lógica subdividir más. Estos casos de uso después se han implementado bottom-up, empezando por los casos de uso atómicos y subiendo la escalera lógica.

Para hacer más comprensible este documento, los casos de uso se presentarán aquí en orden inverso a como se desglosaron - bottom-up, empezando con los elementos atómicos, aproximadamente de la forma en que se implementaron.

A. Casos de uso de funcionamiento básico

Por casos de usos de funcionamiento básico entendemos las acciones mínimas necesarias para interactuar con la interfaz de Google.

A1. Abrir conexión con Google

Nombre	Abrir conexión con API Google
Descripción	Comprueba si la API de Google es reachable, y en caso afirmativo, usa las credenciales guardadas para abrir una conexión y guardarla en un objeto Client.
Actores	Sistema GoogleAdsProject, API Google
Precondiciones:	- Archivo de credenciales existe en la carpeta apropiada
Postcondiciones:	- Conexion con Google Ads API abierta - Objeto Client creado con la conexión y credenciales
Flujo esperado:	1. Seleccionar fichero de credenciales 2. Leer fichero de credenciales 3. Seleccionar cuenta customer de Google 4. Construir objeto Google Oauth 5. Enviar objeto a Google 6. Recibir Response correcta y generar Client

Extensiones/errores	2A. Fichero de credenciales no existe - 2A.1: Devolvemos mensaje de excepción FileNotFound 6A. No se puede contactar con Google - 6A.1: Devolvemos mensaje de excepción de servicio no encontrado

A2. Obtener objetos Ads de Google

Este caso de uso es realmente múltiples casos de uso, ya que técnicamente, cada objeto de Ads requeriría un caso de uso: un caso de uso Guardar AdGroup, Guardar Ad... Dado que todos tienen exactamente el mismo flujo, cambiando solo los parámetros requeridos y el tipo de la respuesta, se han condensado en un solo caso de uso.

Nombre	Obtener Objetos Ads
Descripción	Obtiene uno o multiples elementos de Ads haciendo una petición a la API.
Actores	Sistema GoogleAdsProject, API Google
Precondiciones:	- Conexión con Google abierta mediante caso de uso A1 - Conjunto de parámetros a buscar o condiciones
Postcondiciones:	Uno o más objetos del tipo apropiado con información de la API
Flujo esperado:	1. Construir query GQL a partir de los parámetros 2. Crear request con la query y el tipo de objeto a obtener 3. Enviar request mediante el Client 4. Recibir colección de rows de Google API 5. Parsear la colección de rows a objetos del tipo requerido
Extensiones/errores	4A. Request rechazada por Google API - 4A.1: Obtenemos todos los mensajes de error de la respuesta de Google - 4A. 2: Devolvemos mensaje de excepción con listado de mensajes de error.

A3. Guardar objetos Ads en Google

Similar al anterior, este caso de uso condensa todos los casos de uso idénticos para las diferentes entidades de Google.

Nombre	Guardar Objetos Ads
Descripción	Envía uno o más objetos Ads de un único tipo a Google Ads.
Actores	Sistema GoogleAdsProject, API Google
Precondiciones:	- Conexión con Google abierta mediante caso de uso A1 - Uno o más objetos Ads del mismo tipo
Postcondiciones:	- Nuevas entidades añadidas a Google Ads - Listado con los nuevos resourceNames de las entidades añadidas.
Flujo esperado:	1. Para cada objeto a guardar: a. Crear una entidad Operation b. Si el objeto no tiene resource name, marcar el tipo de operación como CREATE 2. Crear request y añadir las Operaciones creadas en 1 3. Enviar request mediante el Client 4. Recibir colección de resourceNames para las entidades 5. Devolver elemento Response con los resourceNames y posibles errores
Extensiones/errores	1bA: El objeto sí tiene resource name - 1bA.1: Marcamos el tipo de operación como UPDATE - 1bA.2: Marcamos en la operación los campos específicos a actualizar 4A. Request rechazada por Google API - 4A.1: Obtenemos todos los mensajes de error de la respuesta de Google - 4A. 2: Devolvemos mensaje de excepción con listado de mensajes de error.

B. Lectura de elementos

B1. Obtener porcentaje de disponibilidad de Grupos chollo

Nombre	Obtener porcentaje de disponibilidad
Descripción	Obtiene el porcentaje de disponibilidad restante de uno o más grupos chollo.
Actores	Sistema Esquiades, Sistema Buscounchollo
Precondiciones:	Listado de elementos Grupo
Postcondiciones:	Listado de parejas Grupo/porcentaje
Flujo esperado:	1. Obtener los ids de todos los grupos del listado 2. Enviar listado a Buscounchollo 3. Para cada id: a. Obtener grupo a partir de identificador b. Obtener información de disponibilidad original c. Obtener información de ocupación actual d. Calcular porcentaje de actual sobre original e. Asociar porcentaje calculado a id 4. Enviar listado resultante a Esquiades
Extensiones/errores	2A. Error de comunicación (Buscounchollo no disponible, error de ssl…) - 2A.1: Extraer mensaje de error - 2A.2: Enviar email de error a sysadmin - 2A.3: Devolver excepción con mensaje de error extraído.

B2. Extraer json a partir de Grupo chollo

Nombre	Parsear Grupo a Json
Descripción	Para evitar tener que enviar entidades Grupo, que no son serializables, en comunicaciones con GoogleAdsProject, este caso de uso prepara un json con solo los parámetros necesarios, listo para enviar en una petición.
Actores	Sistema Esquiades
Precondiciones:	Entidad Grupo

Postcondiciones:	Mensaje Json
Flujo esperado:	1. Obtener identificador Ads del Grupo 2. Obtener parámetros sencillos 3. Calcular texto de alojamiento/noches 4. Calcular status en que habrá de estar el anuncio: a. Comprobar fin de plazo y categoría b. Comprobar fechas disponibles c. Enviar petición POST a Buscounchollo para que ejecute el caso de uso B1 para obtener porcentaje d. Combinar resultados de a, b, y c para obtener un Status según las reglas de negocio 1-4 5. Añadir patrones de keywords 6. Construir Json con parámetros, textos, y status
Extensiones/errores	1A: El Grupo todavía no tiene identificador para Ads - 1A.1: Calcular nuevo identificador Ads para el Grupo (ver regla de negocio 5, debajo) - 1A.2: Asignar identificador Ads al Grupo - 1A.3: Volver a 1 4cA: Error de comunicación al intentar - 4cA.1: Añadimos mensaje de error - 4cA.2: Se ignora el porcentaje al hacer el calculo en 4d

Reglas de negocio específicas para este caso de uso:

1. Si quedan menos de 10 horas para que acabe el plazo, el anuncio deberá ser cerrado.
2. Si no hay fechas disponibles en la semana siguiente, el anuncio debe ser pausado.
3. Si queda menos de un 15% de disponibilidad, el anuncio deberá ser pausado
4. Si el chollo es de categoría “Top Chollo”, la regla 1 se ignorará.
5. Para crear identificadores Ads, el orden de prioridad será el siguiente:
    a. Si el grupo tiene hotel asociado, se usará el id de hotel.
    b. Si no, se intentará usar el id de región.
    c. Si no tiene región, el chollo es internacional, y se identificará por país (nunca habrá más de un chollo simultáneo internacional para cada país).

B3. Construir lista de entidades GoogleAdsEntity a partir de Json

Nombre	Construir GoogleAdsEntityList
Descripción	En este caso de uso se lee un json que incluyo uno o multiples jsons obtenidos a partir de B2 y se convierten en entidades GoogleAdsEntity, que se validan y añaden a un GoogleAdsEntityList. Este caso de uso se presenta separado porque considero que conforme aumentemos el scope del proyecto, será necesario ejecutar esto en multiples flujos.
Actores	Sistema GoogleAdsProject
Precondiciones:	Tenemos un array json resultado de haber ejecutado el caso de uso B2 N veces para generar Json de parámetros
Postcondiciones:	GoogleAdsEntityList conteniendo N entidades GoogleAdsEntity válidos
Flujo esperado:	1. Construir array a partir de json 2. Generar nuevo GoogleAdsEntityList 3. Para cada línea: a. Generar nuevo GoogleAdsEntity b. Leer json y asignar valores a GoogleAdsEntity c. Validar GoogleAdsEntity d. Añadir GoogleAdsEntity a lista
Extensiones/errores	3cA: Valores no válidos - 3cA.1: Eliminar GoogleAdsEntity, - 3cA.2: pasar a siguiente entidad (no se añade a la lista, no se crea mensaje de error)

C. Creación y modificación de elementos Google Ads

Debido a la naturaleza de Google Ads, crear un “anuncio” completo en la API es un proceso con múltiples pasos. Los diferentes elementos se crean en casos de uso separados.

C1. Crear AdGroups a partir de GoogleAdsEntity

Nombre	Crear AdGroups desde GoogleAdsEntity
Descripción	Crea AdGroups vacíos pero con todos los parámetros a partir de entidades GoogleAdsEntity

Actores	Sistema GoogleAdsProject, Google API
Precondiciones:	Ejecutado caso de uso B3 para obtener lista de entidades. Ejecutado caso de uso A1 para abrir conexión
Postcondiciones:	En Google: Creadas/actualizadas entradas AdGroup En GoogleAdsProject: - Guardados nuevos AdGroupReference en la base de datos - GoogleAdsEntityList actualizada con resource names de las entradas creadas en Google
Flujo esperado:	1. Buscar AdGroupReferences guardadas relacionadas con las entidades en la lista 2. Actualizar GoogleAdsEntityList con valores de los AdGroupReference relacionados 3. Generar nuevos AdGroups 4. Ejecutar caso de uso A2, usando como parametro los resource names de aquellos AdGroups cuyas referencias tuvieran resource names, ya que estos anuncios ya existen en Google API. 5. Contrastar si estos elementos necesitan ser actualizados 6. Unir todos los elementos que necesiten ser creados o actualizados y ejecutar caso de uso A3. 7. Persistir nuevas entidades AdGroupReference para los AdGroups completamente nuevos 8. Devolver respuesta OK
Extensiones/errores	7A: Algunos elementos rechazados - 7A.1: Desglosar mensaje de error por causas - 7A.2: Añadir causas de error a respuesta y marcarla como KO. - 7A.3: Eliminar elementos que no se han podido crear en API de la lista - 7A.4: Continuar con el paso 7 del flujo normal

C2. Crear AdGroupAds a partir de GoogleAdsEntity

Nombre	Crear AdGroupAds desde GoogleAdsEntity
Descripción	Crea AdGroupAds a partir de entidades GoogleAdsEntity, asociándolos con los AdGroups apropiados y
Actores	Sistema GoogleAdsProject, Google API

Precondiciones:	Ejecutado caso de uso B3 para obtener lista de entidades. Ejecutado caso de uso A1 para abrir conexión Ejecutado caso de uso C1 para asegurar que hay AdGroups para los nuevos Ads y AdGroupAds
Postcondiciones:	En Google: Creadas nuevas entradas AdGroupAd, Ad, AdInfo En GoogleAdsProject: GoogleAdsEntityList actualizada
Flujo esperado:	1. Obtener los patrones de anuncios 2. Para cada elemento de GoogleAdsEntityList: a. Generar un Ad por cada patron obtenido en paso 1 b. Generar un AdGroupAd para cada elemento creado en 2a 3. Contrastar si estos elementos necesitan ser actualizados 4. Separar elementos según si necesitan ser creados con AdGroupAd, o solo necesitan actualizar el Ad 5. Ejecutar caso de uso A3 sobre los elementos de paso 4 6. Devolver respuesta OK
Extensiones/errores	5A: Caso de uso A3 devuelve excepción - 5A.1: Desglosar mensaje de error por causas - 5A.2: Añadir causas de error a respuesta KO. - 5A.3: Devolver respuesta KO

C3. Crear AdGroupCriterion a partir de GoogleAdsEntity

Nombre	Crear keywords a partir de GoogleAdsentity
Descripción	Crea un conjunto de AdGroupCriterion de tipo Keyword a partir de un GoogleAdsEntity
Actores	Sistema GoogleAdsProject, Google API
Precondiciones:	Ejecutado caso de uso A1 para abrir conexión con Google Ejecutado caso de uso B3 para obtener GoogleAdsEntity Ejecutado caso de uso C1 para crear AdGroups bajo los que colgar las nuevas keywords
Postcondiciones:	AdGroupCriterions de tipo Keyword creados en Google

Flujo esperado:	1. Comprobar que GoogleAdsEntity pertenece a un AdGroup nuevo 2. Generar textos keywords de tipo amplio 3. Generar textos keywords de tipo estricto 4. Generar textos keywords de coincidencia 5. Para cada elemento obtenido en 2,3,4, generar un nuevo AdGroupCriterion de tipo Keyword 6. Ejecutar caso de uso A3 sobre los elementos obtenidos en 5 7. Devolver respuesta OK
Extensiones/errores	1A. GoogleAdsEntity es de anuncio ya existente - 1A.1: Crear respuesta vacía - 1A.2: Terminar caso de uso 6A: Caso de uso A3 devuelve excepción - 6A.1: Desglosar mensaje de error por causas - 6A.2: Añadir causas de error a respuesta KO. - 6A.3: Devolver respuesta KO

C4. Crear Anuncios completos a partir de GoogleAdsEntityList

Nombre	Crear Anuncios completos
Descripción	Este es el caso de uso "intuitivo" – crea un anuncio completo (es decir, un AdGroup con *todas* sus entidades relacionadas) por cada GoogleAdsEntity en una lista y persiste las referencias necesarias.
Actores	Sistema GoogleAdsProject, Google API
Precondiciones:	GoogleAdsEntityList

Postcondiciones:	Para cada elemento de la lista, existirá lo siguiente:
	- En Google:
	  o Un grupo de anuncios AdGroup
	  o N anuncios Ad, uno por cada plantilla de anuncio existente
	  o Una entidad de relación AdGroupAd por cada Ad que relaciona el AdGroup y el Ad
	  o N AdGroupCriterions relacionados con el AdGroup
	- En GoogleAdsProject: un AdGroupReference persistido en DB vinculando el AdGroup y el Grupo del GoogleAdsEntity
	
	Se devuelve también una respuesta con la combinación todas las respuestas encontradas durante la ejecución

Flujo esperado:	1. Ejecutar A1 2. Ejecutar C1 3. Ejecutar C2 4. Ejecutar C3 5. Construir respuesta a partir de las respuestas OK o KO de los pasos 2, 3, 4 6. Devolver respuesta construida
Extensiones/errores	1A. Caso de uso A1 devuelve excepción - 1A.1: Obtener mensaje de excepción - 1A.2: Construir respuesta de error - 1A.3: Interrumpir caso de uso y devolver respuesta de error

C5. Pausar anuncios no activos

Nombre	Pausar Anuncios No Activos
Descripción	Pausa todos los anuncios de una campaña que estén actualmente subidos y activos pero que no se encuentren referenciados en la GoogleAdsEntityList recibida por parámetro
Actores	Sistema GoogleAdsProject, Google API
Precondiciones:	GoogleAdsEntityList actualizada con todos los resource names (es decir, habiendo pasado por el caso de uso C1 o similares) Ejecutado caso de uso A1.

	Recibe además el id de la campaña sobre la que se quiere actuar como parámetro
Postcondiciones:	Todos los anuncios de la campaña cuyos resource names no se encuentren en la GoogleAdsEntityList estarán pausados en el sistema de Google. Se devuelve un log con los resource names de los anuncios pausados y posibles errores.
Flujo esperado:	1. Ejecuta caso de uso A2 para obtener todos los AdGroups (que no estén ya pausados) de la campaña 2. Obtiene todos los resource names de las entidades en GoogleAdsEntityList 3. Contrasta los elementos obtenidos en 1 y 2 y obtiene los resource names de la diferencia 4. Para cada resource name obtenido en 3, creamos un AdGroup "esqueleto" con solo los campos de resource name y status = paused 5. Ejecuta el caso de uso A3 sobre la colección 6. Crea log a partir de la respuesta obtenida en paso 5 7. Devuelve log
Extensiones/errores	1A: Caso de uso A2 devuelve excepción - 1A.1: Creamos log respuesta con mensaje explicando la causa del error - 1A.2: Saltamos a paso 7

Nota de diseño: Si bien es cierto que a nivel de metáfora tendría más sentido enviar una lista de anuncios a pausar, dada la naturaleza de lo que se quiere hacer (pausar todos los anuncios de grupos no activos automáticamente), y que durante el caso de uso padre ya estamos recibiendo la lista de grupos activos, se ha considerado más práctico diseñar el caso de uso de esta forma.

D. Casos de uso generales

D1. Ejecutar actualización de campaña de Buscounchollo

Este es el caso de uso "real", el que realmente se ejecuta desde "fuera", por así decir. Todos los casos de uso anteriores se llaman en este, de forma concéntrica.

Nombre	Actualizacion Completa Buscounchollo
Descripción	Lee todos los grupos chollo actualmente activos y realiza todas las actualizaciones necesarias para sincronizar la campaña

Actores	Sistema Esquiades, Sistema Buscounchollo, Sistema GoogleAdsProject, Google API
Precondiciones:	Conjunto de Grupos activos Campaña de Google Ads preexistente
Postcondiciones:	Todas las entidades en el sistema de Google se han sincronizado con el estado actual de los grupos chollo
Flujo esperado:	1. Sistema envía señal de que se requiere actualizar Google Ads. 2. Esquiades obtiene una lista de Grupos chollo actualmente activos 3. Esquiades ejecuta el caso de uso B2 sobre cada Grupo obtenido en 2 4. Esquiades envía una petición POST a GoogleAdsProject con el array de Jsons obtenido en 3, y la campaña y customer de Buscounchollo 5. GoogleAdsProject ejecuta el caso de uso A1. 6. GoogleAdsProject ejecuta el caso de uso B3 7. GoogleAdsProject ejecuta el caso de uso C4 8. GoogleAdsProject ejecuta el caso de uso C5 9. Creamos log de respuesta unificado a partir de las respuestas obtenidas en 7 y 8 10. GoogleAdsProject devuelve el log de respuesta a Esquiades 11. El log de respuesta se analiza por si se requieren acciones adicionales. 12. Sin acciones adicionales, el log se envía de vuelta al sistema para archivado.
Extensiones/errores	4A: No se puede contactar con GoogleAdsProject - 4A.1: Creamos log respuesta con mensaje informando del error de conexión - 4A.2: Saltamos directamente a paso 11 5A: Caso de uso A1 devuelve excepción - 5A.1: Creamos log respuesta con mensaje informando del error al intentar establecer conexion con Google - 5A.2: Saltamos a paso 10 12A: El log contiene errores que requieren acciones adicionales - 12A.1: Se crea un correo a partir del log

	- 12A.2: Se envía el correo a la dirección de correo de control de errores de la empresa - 12A.3: Se envía el log al sistema para archivarlo

7. Diagramas de clases

Este proyecto es un tanto peculiar en tanto se han creado realmente muy pocas clases entidad nuevas. Casi todo el proyecto funciona mediante servicios, los cuales actúan o bien sobre las clases entidad ya proporcionadas por Google en la librería de la API (cuyos diagramas no pegaré enteros aquí, porque están mucho más legibles en la documentación en formato web), o alguna de las clases entidad y lista de los dos diagramas siguientes:

7.1 Diagrama GoogleAdsEntity

La clase GoogleAdsEntity es prácticamente un DTO, no contiene ninguna lógica propia, pero es la clase más importante del flujo del programa. Se monta a partir de los parámetros obtenidos en el caso de uso B3, y almacena todos los datos necesarios para crear todos los elementos de un "anuncio" – el grupo, los anuncios, y las keywords

Los servicios que se llaman en el caso de uso D1 operan sobre un GoogleAdsEntityList, y sobre las GoogleAdsEntity contenidas en dicha lista.

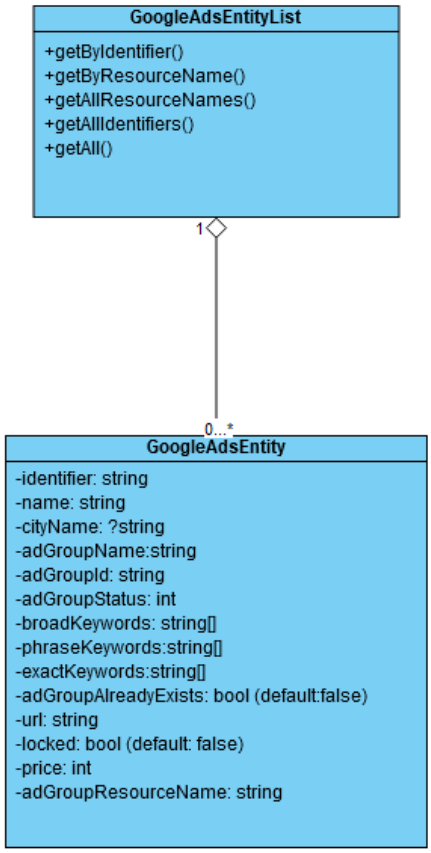

Ilustración 9: GoogleAdsEntity

7.2 Diagrama AdGroupReference

AdGroupReference no es más que una clase relación entre dos clases externas al proyecto GoogleAdsProject, con algún campo adicional.

Para explicar la necesidad que requirió esta clase: el problema que se nos presentaba es que en Esquiades y Buscounchollo lo que tenemos son entidades Grupo (que es como se llama al chollo en la base de datos de la empresa), pero en el sistema de Google lo que tenemos son entidades AdGroup, que representan grupos de anuncios con keywords y similares. En ninguno de los dos casos podíamos añadir un campo de identificación, pero necesitábamos alguna forma de que el proyecto GoogleAdsProject pudiese saber a qué Grupo de nuestro sistema corresponde cada AdGroup subido en el sistema de Google.

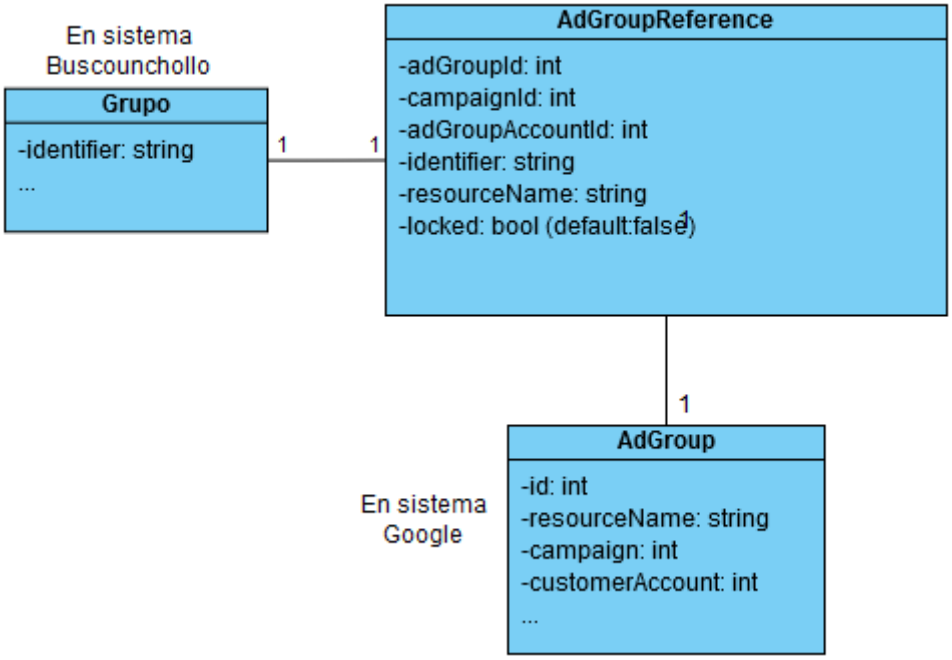

Ilustración 10: AdGroupReference

Grupo y AdGroup tienen muchos más fields, pero a nivel de diagrama lo que nos interesa son solo los campos que identifican con exactitud cada elemento, ya que este objeto mayormente existe para poder mantener una relación 1 a 1 entre dos entidades en las cuales no podemos poner claves directamente.

(El parámetro locked se añadió después, ya que Marketing quería la opción de bloquear algunos anuncios para que no se actualizasen)

Es importante hacer notar que la idea de la clase AdGroupReference es no estar particularmente limitada al Grupo, *per se*. Todo lo que se requiere es que la clase con la que estemos uniendo **tenga un identificador Ads único**. De esta forma, si en el futuro queremos hacer anuncios basados en otras entidades, AdGroupReference nos seguirá valiendo como clase puente.

7.3 Auxiliar: Entidades de Google (resumidas)

Como he mencionado, las relaciones detalladas de las clases de Google (con los enums de los diferentes campos, las relaciones con elementos externos, y demás) están en la documentación de Google, pero para hacer más fácil de comprender las descripciones posteriores, presento aquí un diagrama simplificado (sin copiar todos los campos de cada elemento, que son muchos) de la estructura de clases de los anuncios en Google en las **principales** clases que tocamos: AdGroup, AdGroupAd, Ad, AdGroupCriterion, AdInfo, así como las clases generales de las que cuelgan, Campaign y Customer.

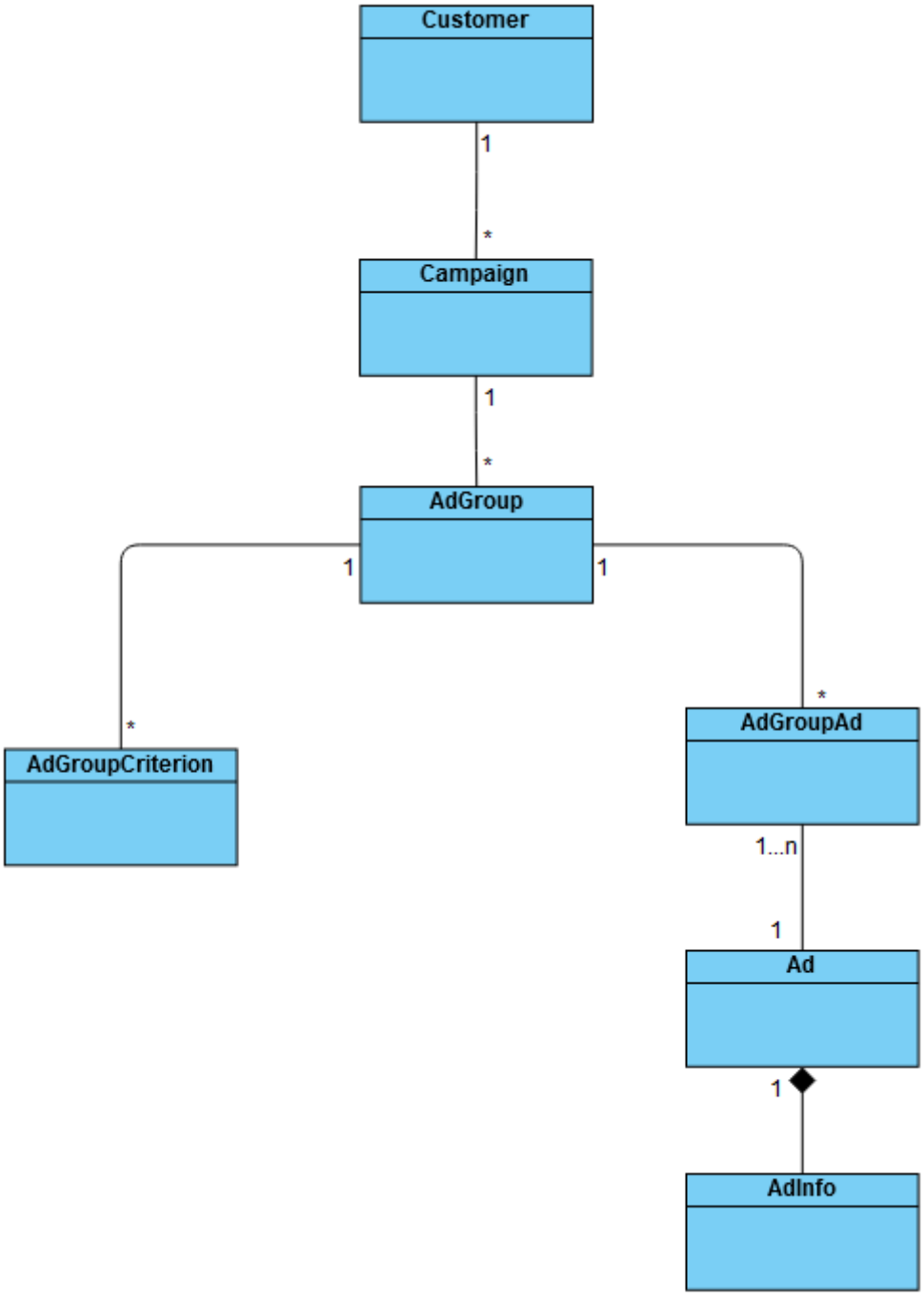

Ilustración 11: Entidades de Google principales

He querido mostrar este diagrama también para hacer notar las diferencias entre las metáforas ofrecidas por Google, y la relación real entre estas clases.

Por ejemplo, podemos ver que si bien la metáfora ofrecida es que el anuncio es un elemento único, realmente **crear un anuncio bajo un AdGroup son tres entidades**: un Ad (entidad principal del anuncio), un AdInfo (que contiene los textos, enlaces, y similares), y, peculiarmente, un AdGroupAd que relaciona el AdGroup con el Ad, lo que parece poco intuitivo a partir de la metáfora.

Ahora bien, si el lector se fija en las multiplicidades notará que, si bien un AdGroupAd solo dependerá de un AdGroup y solo tendrá un Ad, pueden existir múltiples AdGroupAds que apunten al mismo anuncio. Es decir, el mismo Ad puede ser parte de múltiples grupos, simplemente creando más AdGroupAds. Esta no es una funcionalidad que hayamos utilizado en este proyecto, pero sí que ha sido necesario tenerlo en cuenta, ya que este diseño hace que la navegabilidad hacia "atrás" sea prácticamente inexistente.

(Este es el motivo por el cual los resource names de los Ads tienen el formato *customers/{customer_id}/ads/{ad_id}* – la ruta del Ad no puede usar el AdGroup, porque pueden ser múltiples AdGroups para un Ad)

8. Arquitectura y flujo

En general, la mayor parte del proceso se lleva a cabo en el nuevo proyecto, pero hay partes que se han implementado en proyectos existentes por simplicidad y acceso a bases de datos. Por otro lado, el programa por necesidad se comunica con una entidad externa, Google. Así pues, podríamos pensar que el flujo de este programa tiene cuatro "nodos" que ejecutan lógica y entre los cuales circulan mensajes:

Esquiades: Al ser el nodo central de la empresa, he ubicado aquí tanto el script que se llama periódicamente para preparar y ejecutar el caso de uso central, como el servicio que parsea los grupos a json de parámetros, ya que este servicio requiere acceso a la base de datos principal de la empresa.

Buscounchollo: Dado que este es el nodo con más carga de usuarios de la empresa, se ha intentado minimizar la cantidad de trabajo que se pide a este nodo. Por tanto, en este nodo solo hemos ubicado el servicio que calcula la ocupación para cada grupo chollo, ya que se requiere acceso a los datos de ocupación que están solo en la BD de Buscounchollo.

GoogleAdsProject: el nodo principal que contiene casi todos los servicios del programa. Es aquí donde se lleva a cabo la mayoría de operaciones. Es el único que se comunica con Google.

Google: El endpoint de Google, que recibe los mensajes que envía GoogleAdsProject y devuelve responses.

En la página siguiente se adjunta un diagrama sencillo de secuencias entre los nodos implicados en el programa, para ayudar a entender el movimiento entre nodos:

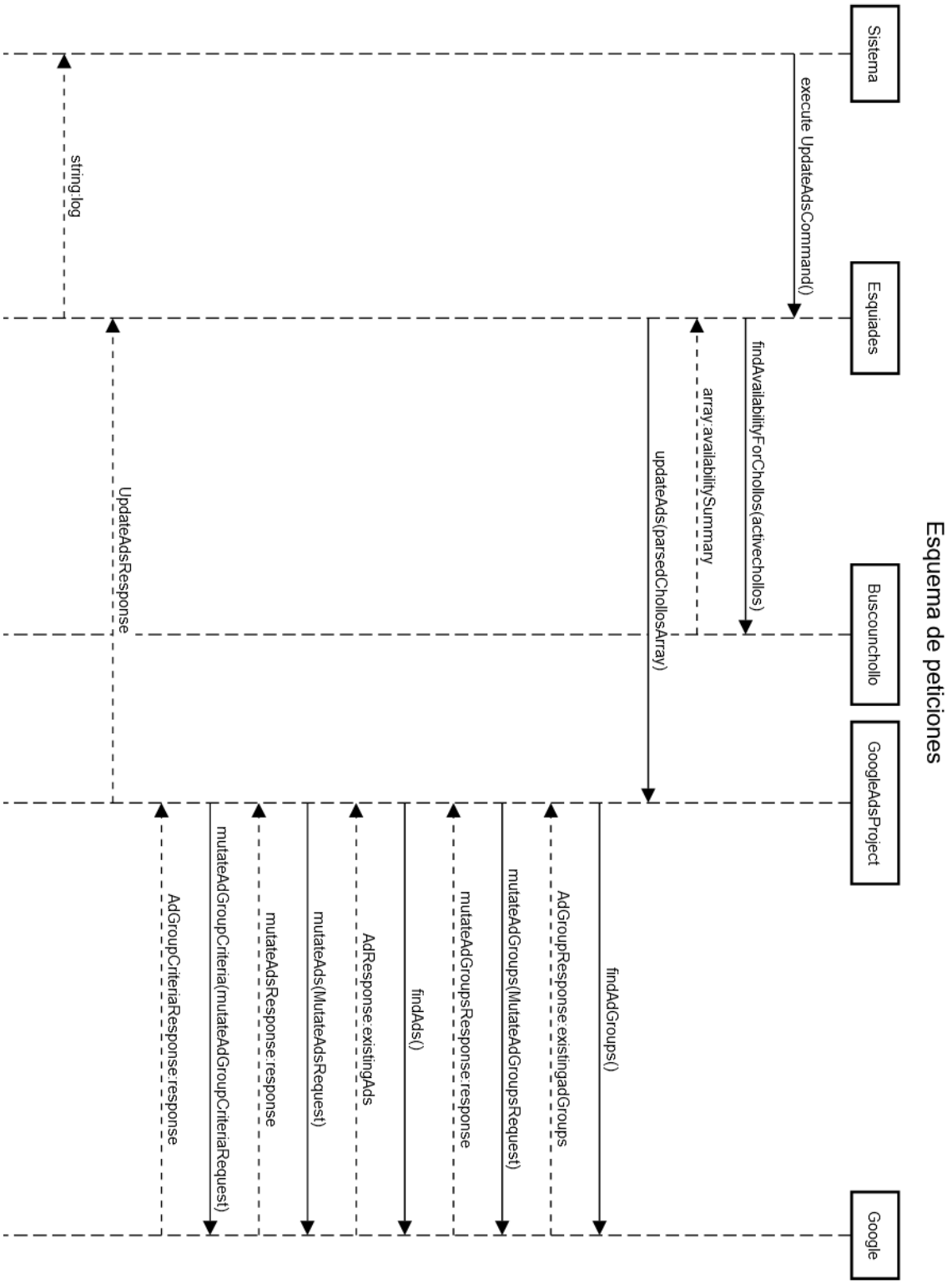

Ilustración 12: Diagrama de flujo general

9. Implementación

9.1 Implementación inicial

Como se ha mencionado en los primeros apartados, implementar este proyecto ha resultado algo más complejo a nivel tecnológico de lo previsto. Por suerte, conté con la ayuda de los sysadmins de la empresa, que me orientaron bastante con los requisitos iniciales.

9.1.1 Creación de nueva base de datos

Para esto, no tuve que hacer yo la configuración. La empresa ya tiene un sistema de contenedores múltiples con varias bases de datos, así que la única cuestión era a que contenedor añadir la nueva base de datos SQL.

Tras hablarlo con el equipo de sysadmins, decidimos que, ya que la base de datos de GoogleAdsProject sería muy pequeña (el diseño original había incluido dos tablas, el diseño final solo requirió una), pero el contenedor de la base de datos de Esquiades ya estaba muy cargado, pondríamos la nueva base de datos en el contenedor de mysql-services, que no tiene muchos recursos de espacio porque es básicamente para servicios auxiliares como el mío, que necesitan menos de cinco tablas.

Por tanto, todo lo que se requirió fue crear un nuevo conector en el contenedor de mysql services, añadir una nueva base de datos, y crear la tabla para los AdGroupReferences dentro de la base de datos:

```
create table ad_group_reference
(
      identifier varchar(255) not null,
      ad_group_id bigint not null,
      campaign_id bigint not null,
      resource_name varchar(255) not null
            primary key,
      account_id bigint not null,
      locked tinyint(1) default 0 not null
);
```

9.1.2 Creación de nuevo Docker

Dado que la empresa ya tiene un sistema de Dockers montado, lo que hizo falta fue configurar un nuevo docker.

Esto requiere dos cosas: una imagen que montar en el docker, y un archivo de configuración docker-compose.yml con las características necesarias para que docker pueda crear el nuevo entorno.

A nivel de imagen, tuvimos la ventaja de que la empresa ya tenía imágenes prediseñadas, así que en vez de partir de cero, lo que hicimos fue duplicar uno de los Dockerfiles (que son básicamente recetas para crear imágenes docker) que se usaron para crear las imágenes existentes y hacer algunas modificaciones, resultando en el siguiente dockerfile:

```
FROM php:8.0.3-apache-buster
EXPOSE 22 25 80 443

# Add ubuntu user and set a password for root user
RUN adduser --disabled-password --gecos "" ubuntu
RUN echo "root\nroot" | (passwd root)

RUN apt-get update && apt-get install -y jq zip unzip nano git
python3-pip g++ zlib1g-dev msmtp-mta \
        libcurl4-openssl-dev libpng-dev libxml2-dev libzip-dev
libicu-dev libc-client-dev libkrb5-dev npm openssh-server \
    # Prerequisites instalation
    &&  apt-get clean && apt autoremove -y && rm -rf
/var/lib/apt/lists/* /tmp/* /var/tmp/* \
    # Delete temporal files
    && sed -i 's/www-data/ubuntu/g' /etc/apache2/envvars
    # Change apache default user to ubuntu

RUN php -r "copy('https://getcomposer.org/installer', 'composer-
setup.php');" \
    &&  php composer-setup.php --version=2.0.12 --install-
dir=/usr/local/bin --filename=composer \
    &&  php -r "unlink('composer-setup.php');" \
    # Composer Installation
    &&  printf "\n" | pecl install xdebug apcu protobuf-3.17.0 grpc
&& docker-php-ext-enable xdebug apcu protobuf grpc \
    # Apcu + Xdebug + Protobuf + grpc Install and enable
    &&  pip3 install awscli
    # AWS CLI installaton

RUN docker-php-ext-configure intl && docker-php-ext-configure imap -
-with-kerberos --with-imap-ssl \
    # PHP extension configuration
    && docker-php-ext-install mysqli pdo pdo_mysql imap gd zip xml
soap intl opcache
    # PHP extensions installation

RUN npm install -g npm@6.14.8 && npm install -g gulp-cli@2.2.0 \
    # Npm + gulp installation
    &&  rm $APACHE_CONFDIR/sites-enabled/* && a2ensite 000-
default.conf && a2enmod rewrite ssl \
    # Apache sites remove and ensite, install mod rewrite
    && rm /var/log/apache2/*
    # Delete apache2 default log files to create valid ones on
service startup

COPY php.ini $PHP_INI_DIR/php.ini
# Copy php.ini to correct folder
COPY project-* /usr/local/bin/
# Copy scripts to call them without path
ADD viajesparatiCA.crt /usr/local/share/ca-certificates
ADD .msmtprc /home/ubuntu/.msmtprc
ENTRYPOINT ["project-entrypoint"]
```

y después entrar a la imagen para instalar los requisitos indicados en el apartado 5.1.3 Necesidades y requisitos.

A nivel de docker-compose.yml, la configuración que creé fue esta:

```
google-ads:
    tty: true
    hostname: google-ads
    image: harbor.viajesparati.local/registry/apache:php8.0
    depends_on:
      - mysql-services
    external_links:
      - mysql-services:db-google-ads
    logging:
      driver: "json-file"
      options:
        max-size: "50m"
        max-file: "1"
    privileged: true
    environment:
      - PROJECT=google-ads
      - EMAIL=${USER}@esquiades.com
      - DISPLAY=:0
    volumes:
      - ../google-ads:/var/www/html:consistent
      - ../.credentials/common:/root/.aws:ro
      - ../.credentials/certificates:/root/certs
    networks:
      full:
        ipv4_address: 10.42.160.109
```

Como puede verse, el docker-compose especifica el nombre del docker, los elementos con los que se conecta (en este caso, solo el servicio de la base de datos), la forma en que se logean los errores, variables básicas de entorno, las definiciones de los volúmenes (incluyendo los volúmenes en los que guardamos credenciales de AWS y similares), y la red a la que se conectará.

Una vez listo esto, el docker está funcional en local.

Para tenerlo funcional en producción, había que subirlo a Amazon ECS.

Esto implica básicamente:

- repetir los pasos anteriores en el servicio de Amazon pero cambiando los parámetros apropiados del docker-compose (que no copiaré aquí porque incluye algún elemento confidencial de la empresa)
- Subir la imagen docker resultante a Amazon Elastic Container Registry.
- Crear el fichero de configuración Google-ads.json que indicará como tendrán que ser los contenedores que se generen.

```
 {
   "taskRoleArn": "arn:aws:iam::REDACTED FOR PRIVACY:role/vpt-
google-ads-ecsTaskExecutionRole",
    "networkMode": "awsvpc",
    "containerDefinitions": [
```

```
      {
        "memoryReservation": 128,
        "name": "google-ads-api_web",
        "image": "replace",
        "essential": true,
        "portMappings": [
          {
            "containerPort": 80,
            "protocol": "tcp"
          },
          {
            "containerPort": 443,
            "protocol": "tcp"
          }
        ],
        "logConfiguration": {
          "logDriver": "awslogs",
          "options": {
            "awslogs-group": "ECS_google-ads",
            "awslogs-region": "eu-west-1",
            "awslogs-stream-prefix": "google-ads"
          }
        },
        "environment": [
          {
            "name": "APACHE2_OPTS",
            "value": ""
          }
        ],
        "memory": 512,
        "cpu": 32,
        "volumesFrom": []
        }
    ],
    "executionRoleArn":
"arn:aws:iam::743936116738:role/ecsTaskExecutionRole"
  }
```

- Registrar el docker en la pipeline de deploys de la empresa en Amazon, para que Amazon sepa que cuando se suba código nuevo hay que hacer la rotación de contenedores.

9.1.3 Creación y configuración del proyecto de Symfony

Una vez que tuvimos un docker funcional, pude por fin crear un nuevo proyecto. El proyecto se ha implementado, como la mayoría de los sistemas de la empresa, sobre el framework Symfony, que facilita enormemente el trabajo de iniciar un proyecto nuevo.

Los pasos a seguir para generar el proyecto Symfony fueron tan simples como:

- Dentro del Docker, ejecutar el comando *composer create-project symfony/skeleton google-ads*. Esto creará el esqueleto del proyecto, preparando las carpetas mínimas y archivos yml de configuración.
- Configurar el fichero composer.json

- Ejecutar *composer require <X>*, donde X es cada uno de los requisitos de librerías composer vistos en 5.1.3 Necesidades y requisitos
- Crear un fichero **.htaccess** en la carpeta /public del proyecto con la configuración que usará el servidor apache que recibirá las peticiones (en mi caso, copié la configuración de otro de los proyectos de la empresa)
- Añadir el código siguiente al archivo services.yaml, para mapear solo los elementos de la versión V8 de la librería de google-ads-php, en vez de las versiones previas (esto no es estrictamente necesario, pero cuadruplicar la cantidad de servicios que el sistema mapea cada vez que se hace deploy de nuevo código aumenta el tiempo de deploy significativamente, como puede suponerse)

```
Google\Ads\GoogleAds\Lib\:
   resource:'../vendor/googleads/google-ads-php/src/Google/
   Ads/GoogleAds/Lib'
   exclude:'../vendor/googleads/google-ads-php/src/Google/
   Ads/GoogleAds/Lib/{V5,V6,V7}'
```

Con esto, ya tuve un esqueleto funcional para el proyecto.

Adicionalmente, para que el proyecto funcionase correctamente, fue necesario añadir más tarde lo siguiente al fichero index.php que hace el request handling de Symfony:

```
if ($_SERVER['APP_DEBUG']) {
    $errorReporting = error_reporting();
    umask(0000);

    Debug::enable();
    error_reporting($errorReporting & ~E_USER_WARNING);
}
```

Básicamente, los mensajes protobuf de google utilizan un sistema de definición que hace que se redefinan constantemente, lo que hace saltar USER_WARNING mientras debugamos el código, y Symfony convierte los USER_WARNING en excepciones que interrumpen el programa. Esto hace imposible ejecutar nada con el debugger de Symfony activo. Este arreglo fuerza a ignorar los USER_WARNING mientras debugamos.

(He querido especificar aquí el problema y su solución por si alguien que lea este proyecto en un futuro decide implementar su propia versión del proyecto, ya que nos llevó una semana encontrar qué era lo que estaba causando esto).

9.1.4 Mapeo del docker en local y produccion

Para que los desarrolladores pudieran hacer pruebas al trabajar sobre el proyecto, lo más sencillo es permitir que hagan peticiones a partir del navegador. Para esto, se añadió la siguiente linea al fichero hosts compartido que usan los equipos de la oficina:

10.42.160.109 google-ads.viajesparati.local

En producción, lo que se hizo fue añadir al DNS de la empresa la dirección google-ads.viajesparati (no voy a copiar y apuntarla aquí, por motivos evidentes) al contenedor de ECS en el que se ubicaría el proyecto.

Una vez terminados todos estos pasos, el proyecto estaba oficialmente en funcionamiento, y pudimos empezar a implementar servicios.

9.2 Implementación del script principal

El script principal es el que lleva a cabo el caso de uso D1. Se encuentra en Esquiades, y utiliza un Command de Symfony (en Symfony, un Command es básicamente un script php accesible desde consola que tiene acceso a los servicios mapeados en Symfony mediante el acceso al contenedor de contexto, así como a un logger preconfigurado). Este command se ejecuta mediante un simple crontab que lanza la instrucción de consola cada hora.

El script básicamente tiene 4 partes:

- Parsing de los grupos. Cogemos todos los chollos activos y vamos iterando, parseando todos los que se puedan. Si faltan parámetros, el parser devolverá nulo, así que logeamos el problema y saltamos al siguiente.

Si un Group no tiene identificador, logeamos el problema y le asignamos uno.

El bucle principal de esta sección es como sigue:

```
foreach ($activeChollos as $cholloGroup) {
            if ($cholloGroup->getSaleType() != 0 || $cholloGroup-
>getSecretHotel() == true) {
                continue;
            }
            if ($identifierCreator-
>checkIfNeedsIdentifier($cholloGroup)) {
                $possibleIdentifier = $identifierCreator-
>createIdentifier($cholloGroup);
                if (isset($possibleIdentifier)) {
                    $cholloGroup-
>setGoogleAdsIdentifier($possibleIdentifier);
                    $cholloRepository->save($cholloGroup);
                    $warningText = "Chollo group with id
".$cholloGroup->getId()." needed a new identifier. System set
identifier to ". $possibleIdentifier;
                    $logger->printMessage($warningText);
                } else {
                    $warningText = "Chollo group with id
".$cholloGroup->getId().' is missing an ads identifier, Skipping.';
                    $logger->printMessage($warningText);
                    continue;
                }
            }

            $parsedGroup = $groupParser-
>parseGroupToAdsArray($cholloGroup, $defaultStatus);

            if ($parsedGroup == null) {
```

```
                $warningText = "Chollo group with id ".$cholloGroup-
>getId()." could not be parsed correctly. Skipping.";
                $warningsToSend .= $warningText."<br>\n";
            } else {
                $cholloGroupsToCreate[$cholloGroup->getId()] =
$parsedGroup;
            }
        }
```

- Actualización de los arrays creados durante el parsing dependiendo de porcentaje de disponibilidad: Esto envía un mensaje al sistema de Buscounchollo para utilizar el servicio GroupRoomAvailabilityGetter. En caso de que Buscounchollo devuelva un error (generalmente por uno de varios posibles problemas de conexión) simplemente no editamos los status y procedemos con los status calculados en el parsing – pero preparamos un mensaje para enviar a correo.

```
try {
            $cholloGroupsToCreate = $statusCalculator-
>checkReimainingAvailabilityAndUpdateStatusIfNecessary($cholloGroupsToCreate);
        } catch (RequestException | ClientException $e) {
            $warningText = 'An error happened while trying to get availability
of groups from BUC. Availability was ignored for this execution. Exception
message: ';
            $warningsToSend .= $warningText.$e->getResponse()->getBody()-
>getContents();
        } catch (GuzzleException $e) {
            $warningText = 'An error happened while trying to get availability
of groups from BUC. Availability was ignored for this execution. Exception
message: ';
            $warningsToSend .= $warningText.$e->getMessage()."<br>\n";
        }
```

(Hacen falta múltiples catches debido a que diferentes tipos de excepción de conexión en php tienen mensajes en posiciones diferentes – algunos necesitan un getMessage(), mientras que en otros hay que hacer un getContents() para conseguir un mensaje de error útil)

- Creación de anuncios: Aquí el command simplemente prepara el array y lo envía a GoogleAdsProject, para que se ejecute el caso de uso C5 allí.

Pese a la complejidad del caso de uso, debido al diseño del programa, los únicos errores posibles son siempre errores de conexión o request no válida, ya que las excepciones dentro del proceso de anuncios ya se están interceptando y convirtiendo en logs. Así que lo que hace esta sección del script es sencillamente lanzar el request, asegurarnos de que no hay errores de conexión, y comprobar si el mensaje de vuelta contiene mensajes de error.

```
$bodyData = [
            'campaignId' => $cholloCampaignId,
            'defaultstatus' => $defaultStatus,
            'activeChollos' => $cholloGroupsToCreate
```

```
        ];

        try {
            $requestService = $this->getContainer()-
>get('admin.application.googleads.google_ads_request_service');
            $result = $requestService->doPost('cholloUpdate',
$bodyData);
            $parseresult = json_decode($result->getBody()-
>getContents(), true);
            $returnText = '';
            if (isset($parseresult['adCreationResponse']['errorMsg'])
&& $parseresult['adCreationResponse']['errorMsg'] != '') {
                $returnText .= "adCreationLog: \n" .
$parseresult['adCreationResponse']['errorMsg']."<br>\n";
            }
            if
(isset($parseresult['adDeactivationResponse']['errorMsg']) &&
$parseresult['adDeactivationResponse']['errorMsg'] != '') {
                $returnText .= "AdDeactivation log: " .
$parseresult['adDeactivationResponse']['errorMsg']."<br>\n";
            }
        } catch (RequestException | ClientException $e) {
            $returnText = "An error happened while posting to google
ads project. Exception message:";
            if ($e->getResponse()) {
                $returnText .= "\nError body: ".$e->getResponse()-
>getBody()->getContents();
            } else {
                $returnText .= "\nError message: ".$e->getMessage();
            }

        } catch (GuzzleException $e) {
            $returnText = "An error happened while posting to google
ads project. Exception message:<br>\n".$e->getMessage();
            if (isset($result)) $returnText .= "\nResponse body:
".$result->getBody()->getContents();
        }
```

(En caso de que haya errores los añadimos inmediatamente a los mensajes que se enviarán en correo, ya que errores aquí requerirán acciones por parte de los usuarios)

- Y, finalmente, tomamos los logs que se han podido crear en los apartados anteriores, y en caso de que no estén vacíos, los enviamos a un correo de informe de errores.

```
//Send mail with possible errors
        $messageText = '';
        if ($warningsToSend != '') {
            $messageText .= "BUC Google Ads report: <br>\nCommand
log:<br>\n"
            .$warningsToSend."\n--------\n";
        }
        if ($returnText != '') {
            $messageText .= "API log: <br>\n".$returnText;
        }
```

```
        if ($messageText != ''
&& !(OriginChecker::check(OriginChecker::IS_DEV))) {
            $headers = "MIME-Version: 1.0" . "\r\n" . "Content-type:
text/html; charset=iso-8859-1" . "\r\n";
            mail('REMOVED FOR PRIVACY', 'BUC Google Ads Report',
$messageText, $headers);
        }
```

Si los logs han estado vacíos, o si estamos en entorno de desarrollo (para evitar llenar el correo de errores cuando los desarrolladores están trabajando y haciendo pruebas), no se envía ningún correo de informe.

9.3 Implementación de servicios

Como se ha mencionado, básicamente toda la lógica de este trabajo está en servicios, así que dedicaré este apartado a explicar un poco los servicios más importantes que se han implementado, sus métodos, parámetros, y lo que hacen, así como fragmentos de código que puedan servir para entenderlos.

Solo explicaremos los servicios “principales”, ya que no considero que añadir aquí todos los servicios auxiliares utilizados para, por ejemplo, convertir objetos de Google en arrays, validar parámetros, reemplazar comodines en strings y limpiarlos para crear textos de keywords, leer ficheros de texto con los schemas json para los anuncios, etc., aporten nada de interés al lector.

Separaremos los servicios según el nodo en el que se encuentran.

9.3.1 Servicios en Esquiades:

GroupsToAdsArrayParser

Descripción

Parsea elementos Group (grupo chollo) a arrays listos para codificar como Json y enviar al enpoint de GoogleAdsProject.

Métodos públicos

`public function parseGroupToAdsArray(Group $group,$defaultStatus): ?array`

Recibe un Group y un status base para el anuncio resultante (en caso de que no se calcule ningun status durante el parsing) y devuelve un array con todos los parámetros necesarios para crear anuncios.

Para facilitar al lector entender la estructura del parsing y los diferentes parámetros que se están obteniendo, este método en particular se ha copiado aquí entero.

```
/**
     * Parses a group to a formatted array for passing to the google
ads API project.
     * If not all mandatory fields are set for the group, returns a
null value.
     * @param Group $group
     * @param $defaultStatus
     * @return array|null
```

```
     */
    public function parseGroupToAdsArray(Group $group,
$defaultStatus): ?array
    {
        try {
            $adStatus = $this->adGroupStatusCalculator-
>calculateBaseStatusForGroup($group) ?? $defaultStatus;

            if ($group->getHotelId() != null && $group-
>getHotelId() != 0) {
                /** @var Hotel $hotel */
                $hotel = $this->hotelRepository->find($group-
>getHotelId());
                if (!$hotel) {
                    return null;
                }
                $name = $hotel->getName();
                $adGroupName = $hotel->getName();
                $destination = $hotel->getCityName();
            } else {
                /** @var Chollo $chollo */
                $chollo = $group->getChollos()->first();
                $countryId = $chollo->getCountryId();
                if ($countryId && !in_array($countryId,
self::COUNTRIES_NO_DESTINATION)) {
                    $country = $this->countryRepository-
>find($countryId);
                    $countryName = $country->getName();
                    $name = $countryName;
                    $adGroupName = $countryName;
                    $destination = $countryName;
                } else {
                    return null;
                }
            }

            $keywords = $this->keywordBuilder->buildKeywords($group,
$name, $destination);

            $result = [
                'identifier' => $group->getGoogleAdsIdentifier(),
                'name' => $name,
                'cityName' => isset($hotel) ? $hotel-
>getCityName() : '',
                'adGroupName' => substr($adGroupName, 0,
self::MAX_HEADLINE_LENGTH),
                'adGroupStatus' => $adStatus,
                'broadKeywords' => $keywords['broadKeywords'],
                'phraseKeywords' => $keywords['phraseKeywords'],
                'exactKeywords' => $keywords['exactKeywords'],
                'topChollo' => $group->getTopGroup(),
            ];
            $result = $this->charsFilter->recursiveFilter($result);

            /* Setting the URL for the ad
            Note: URL needs to be a domain accessible by google's
validator, so this always points to the production url.
```

```
            Setting local.buc address in dev environment causes a
ValidationError partial failure even when uploading test ads*/
            $result['url'] =
'https://www.buscounchollo.com/?ad_destacado=' . $group->getId() .
'&new';
            $result['cholloGroupId'] = $group->getId();

            /* Grabbing data from the first chollo not sold out */
            $primaryChollo = $this-
>findPrimaryCholloForGroup($group);

            $countdownEndString = $primaryChollo->getEndDate() . '
' . $primaryChollo->getEndHour();
            $end = new \DateTime($countdownEndString);
            $result['countdownEnd'] = $end->format('Y/m/d H:i:s');

            $nightsString = $this-
>buildNightsStringFromChollo($primaryChollo);
            $result['nightsString'] = $nightsString;

            $price = ceil($primaryChollo->getSalePrice());
            $result['price'] = $price;

        } catch (Throwable $e) {
            //If any part of the process breaks, we will not have
enough information to build a complete ad,
            //so we return null value so that the command may detect
this is not a valid group ad info set
            return null;
        }

        return $result;
    }
```

AdGroupStatusCalculator

Descripción

Contiene métodos que calculan status apropiados para grupos de anuncios.

Métodos públicos

`public function calculateBaseStatusForGroup(Group $group): int`

Aplica las reglas de negocio mencionadas en el caso de uso B2, con la excepción de la regla 3, para obtener el valor numérico del status que debería tener el anuncio de un grupo específico en el momento actual.

`public function checkReimainingAvailabilityAndUpdateStatusIfNecessary(array $groups): array`

Método que recibe un array de arrays parseados con GroupToAdsArrayParser y edita los campos status de los elementos que necesiten ser pausados por baja disponibilidad.

Devuelve el mismo array parámetro, pero con el status actualizado en los elementos apropiados.

Notas de diseño: Si bien el caso de uso B2 originalmente se diseñó asumiendo que se calcularía todo el status de cada grupo dentro del parsing, para reducir la cantidad de peticiones adicionales, se ha separado la aplicación de la regla de negocio 3 de esta forma. Así solo es necesario hacer una única llamada al endpoint que aloja el servicio GroupRoomAvailabilityGetter, en vez de N llamadas, reduciendo tráfico.

De la misma forma, para reducir carga, se intenta minimizar el peso de la request enviada antes de enviarla:

```
public function
checkReimainingAvailabilityAndUpdateStatusIfNecessary(array
$groups): array
    {
        $groupIdsTocheckAvailability = [];
        foreach ($groups as $groupId => $group) {
            //If a group has already been paused, no need to do more
calculations to check if availability requires it to be paused
            if ($group['adGroupStatus'] == self::ADGROUP_PAUSED) {
                continue;
            }
            $groupIdsTocheckAvailability[] = $groupId;
        }

        if (empty($groupIdsTocheckAvailability)) {
            return $groups;
        }
```

Así, los elementos enviados son simples enteros, y los elementos recibidos son parejas de enteros, y es el servicio el que se encarga de asociarlos y editar los elementos del array $groups si corresponde:

```
foreach ($percentages as $groupId => $percent) {
            if ($percent < self::MINIMUM_AVAILABILITY_PERCENT) {
                $groups[$groupId]['adGroupStatus'] =
self::ADGROUP_PAUSED;
            }
        }
```

9.3.2 Servicios en Buscounchollo

GroupRoomAvailabilityGetter

Descripcion

Recibe ids de Groups, y obtiene porcentajes de disponibilidad restantes para dichos groups.

Metodos públicos

`public function getDayTotalsAvailabilityForMultipleGroups(array $groupIds): array`

Recibe un array de ids numéricos correspondientes a entidades Group, y devuelve un array de parejas de valores [id grupo => porcentaje de disponibilidad restante en el grupo].

`public function getAvailabilityArraysForGroup(int $groupId, $link): int`

Lee todas las tablas de disponibilidad asociadas a los elementos del Grupo, obtiene las fechas disponibles, contrasta valores con las reservas realizadas en los alojamientos asociados, y calcula un porcentaje aproximado de desocupación comparando la disponibilidad inicial con las habitaciones ocupadas.

De nuevo, el diseño del sistema se ha intentado montar con la idea de extensibilidad. El servicio incluye un método que calcula el porcentaje para un grupo particular, y calcula los grupos de uno en uno utilizando getAvailabilityArraysForGroup() en el método principal, porque la pequeña pérdida en eficiencia implicada aquí merece la pena a cambio de tener un método que puede calcular fácilmente para un único grupo que pueda utilizarse en otros desarrollos (por ejemplo, se ha proyectado ya añadir una barra de disponibilidad restante en los chollos en el portal).

9.3.3 Servicios en GoogleAdsProject

GoogleAdsSessionClientBuilder

Descripción

Para simplificar los casos de uso, el proceso de conseguir las credenciales de Google y crear un objeto Client que se pueda utilizar en el flujo se ha sintetizado en un servicio sencillo.

Métodos públicos

`public function build($accountId = null):GoogleAdsClient`

Selecciona las credenciales según el id de cuenta (customer) que se le pase, envía una petición de autenticación a Google Ads, y genera un objeto GoogleAdsClient para utilizar en otras clases. Si no se le pasa ningún id, utiliza credenciales de la cuenta central.

```
public function build($accountId = null):GoogleAdsClient
    {
        $oAuth2Credential = $this->oAuth2TokenBuilder
            ->withClientId($this->oauthClientId)
            ->withClientSecret($this->oauthClientSecret)
            ->withRefreshToken($this->oauthRefreshToken)
            ->build();

        $adminAccount = $this->googleAdsAccountsGetter-
>getFromTag('admin');
        $this->googleAdsClientBuilder
            ->withLinkedCustomerId($adminAccount)
            ->withDeveloperToken($this->googleAdsDevToken)
            ->withOAuth2Credential($oAuth2Credential);

        if ($accountId) {
            $this->googleAdsClientBuilder-
>withLoginCustomerId($accountId);
        } else {
            $this->googleAdsClientBuilder-
>withLoginCustomerId($adminAccount);
        }
        $googleAdsSession = $this->googleAdsClientBuilder->build();
        return $googleAdsSession;
    }
```

Repositorios

Generalmente, Doctrine interactúa con entidades persistidas mediante **repositorios**: clases que permiten obtener entidades de un cierto tipo y persistir estas mismas entidades.

Dado que el objetivo de este proyecto era crear una base sobre la cual se pudiera continuar iterando, se han generado los servicios que se encargan de los casos de uso A2 y A3, obtener entidades y guardar entidades, como si fueran repositorios Doctrine normales – excepto que en vez de guardar o leer de base de datos, guardan y leen de la API externa. De esta forma, la idea es que para futuras extensiones, estos casos de uso sean transparentes para los programadores, y que simplemente utilicen los métodos findBy() y save() como si fueran entidades normales.

Por esto, creo que es más útil explicar aquí el diseño general de los repositorios, que simplemente mostrar aquí sus métodos y signatures.

Se han creado 4 repositorios primarios:

- AdGroupRepository
- AdGroupAdRepository
- AdRepository
- AdGroupCriteriaRepository

Todos los cuales tienen al menos un método save($entityCollection) y extienden de una clase abstracta GoogleAdsOperationsRepository.

Repositorio padre: GoogleAdsOperationsRepository

Descripción

Esta clase es abstracta. Simplemente contiene algunos métodos compartidos entre los repositorios para evitar código repetido: el método que coge el cliente creado en el caso de uso A1, el método que lanza la llamada…

Pegaré aquí el método compartido que posiblemente sea el más ejemplar a nivel de diseño: findByCampaign()

```
protected function findByCampaign(int $campaignId, string
$tableName, array $parameters = [], array $fieldsToGet =
self::ALL_FIELDS): PagedListResponse
    {
        $whereStatement = " WHERE campaign.id = ". $campaignId;
        if (sizeof($parameters) > 0) {
            foreach ($parameters as $paramField => $paramValue) {
                if (is_array($paramValue)) {
                    $items = array_map(function ($item) {return
addslashes($item); }, $paramValue);
                    $whereStatement .= ' AND '.$paramField." IN
('".implode("', '", $items)."')";
                } else {
                    $whereStatement .= ' AND '. $paramField . ' =
' . $paramValue;
                }
            }
```

```
        }

        $query = 'SELECT '
            . self::buildQueryFields($tableName, $fieldsToGet)
            .'FROM '.$tableName
            .$whereStatement;

        $client = $this->googleAdsSessionBuilder->build()-
>getGoogleAdsServiceClient();
        $response = $client->search($this->accountId, $query);

        return $response;
    }
```

Dado que la campaña sobre la que operaremos se ha decidido al principio del flujo, todas las operaciones de los repositorios hijos se harán sobre una campaña específica. Por tanto, todos los findBy($params) de los repositorios hijos acabarán pasando por aquí.

La idea es que al método se le pasa la campaña, el nombre de la tabla, un array de parámetros condicionales, y un array de los campos a buscar, siendo estos cuatro parámetros, cosas que o bien pasa el usuario o que las clases hijas tienen, y el método genera una query de GQL y la lanza usando el método search() del cliente Google.

¿Qué quiero decir con que las variables las tienen las clases hijas? Veamos un ejemplo, con la clase AdGroupRepository:

```
class AdGroupRepository extends GoogleAdsOperationsRepository
{
    const ALL_FIELDS = [
        'ad_group.resource_name',
        'ad_group.status',
        'ad_group.type',
        'ad_group.ad_rotation_mode',
        'ad_group.url_custom_parameters',
        'ad_group.explorer_auto_optimizer_setting.opt_in',
        'ad_group.display_custom_bid_dimension',
        'ad_group.effective_target_cpa_source',
        'ad_group.effective_target_roas_source',
        'ad_group.labels',
        'ad_group.id',
        'ad_group.name',
        'ad_group.base_ad_group',
        'ad_group.tracking_url_template',
        'ad_group.campaign',
        'ad_group.cpc_bid_micros',
        'ad_group.cpm_bid_micros',
        'ad_group.target_cpa_micros',
        'ad_group.cpv_bid_micros',
        'ad_group.target_cpm_micros',
        'ad_group.target_roas',
        'ad_group.percent_cpc_bid_micros',
        'ad_group.final_url_suffix',
        'ad_group.effective_target_cpa_micros',
        'ad_group.effective_target_roas'
    ];
```

```
    protected $mutateOperationName = 'MutateAdGroups';
    protected $clientType = 'AdGroupServiceClient';
    protected $returnClassName = 'AdGroup';
    const TABLE_NAME = 'ad_group';
```

Cada una de las hijas tiene estas variables: el nombre de la tabla, las columnas de dicha tabla, el nombre de la entidad sobre la que opera el repositorio, y los nombres de cliente y operación.

De esta forma, expandir el número de repositorios conforme queramos poder trabajar con más entidades es sencillo: tan simple como extender GoogleAdsOperationsRepository, añadir estas variables, y crear un nuevo método save(), y ya estamos listos para operar.

El método save() es importante: así como los métodos findByX() pueden hacerse genéricos, debido a que las queries SELECT son iguales para todos simplemente cambiando los campos, el problema es que las clases que hacen los INSERTS y UPDATES, así como las respuestas de estas operaciones, **no son iguales para las diferentes entidades**. Por tanto, cada repositorio ha de implementar su propio método save().

Las reglas según las cuales se han construido todos los métodos save() son las siguientes:

public function save($entities): {SpecificEntityResponseType}

save() debe aceptar un array de entidades del tipo con el cual opera el repositorio

save() debe devolver una respuesta del tipo específico relacionado con la entidad: MutateAdGroupsResponse, MutateAdsResponse…

save() debe crear operaciones de mutación para cada elemento del array: operación CREATE si la entidad no tiene asignado un resource name, UPDATE en caso contrario

Veamos un ejemplo, de nuevo en AdGroupRepository:

```
public function save($adGroups): MutateAdGroupsResponse
    {
        if (empty($adGroups)) {
            return new MutateAdGroupsResponse();
        }
        $operations = [];
        /** @var AdGroup $adGroup */
        foreach ($adGroups as $adGroup) {
            $operation = new AdGroupOperation();
            if ($adGroup->getResourceName()) {
                $operation->setUpdate($adGroup);
                $updateMask = FieldMasks::allSetFieldsOf($adGroup);
                $operation->setUpdateMask($updateMask);
            } else {
                $operation->setCreate($adGroup);
            }

            $operations[] = $operation;
        }
```

```
        $options = [
            'partialFailure' => true,
            'responseContentType' =>
ResponseContentType::MUTABLE_RESOURCE
        ];

        return parent::operate($operations, $options);
    }
```

Servicios de creación en masa

Estos servicios reciben una GoogleAdsEntityList, un customer y/o una campaña (para saber dónde ubicar las entidades que creen), y utilizan los generadores, updateCheckers, y repositorios para crear y guardar/actualizar todas las entidades necesarias.

Todos estos servicios devuelven lo mismo: un CreationResponse que contiene los logs de errores que se hayan encontrado al ejecutar las mutaciones.

AdGroupCreator

Descripción

Genera y actualiza AdGroups en Google Ads, y persiste y actualiza entidades AdReference que apunten a dichos AdGroups.

Métodos públicos

`public function create(GoogleAdsEntityList $googleAdsHotelList, int $defaultStatus, $campaignResourceName, $customerId): CreatorResponse`

Este método hace cuatro cosas, en orden: genera un AdGroup para cada entidad en la lista parámetro, utiliza AdGroupUpdateChecker para comprobar si estos anuncios han de ser mandados a Google Ads, envía los AdGroups apropiados a Google Ads, y persiste nuevas entidades AdGroupReference para los nuevos grupos que se hayan creado.

A modo de muestra del esquema conceptual de los Creators, que son los tres bastante similares, copio aquí el código principal de AdGroupCreator.

```
public function create(GoogleAdsEntityList $googleAdsHotelList, int
$defaultStatus, $campaignResourceName, $customerId): CreatorResponse
    {
        $adGroups = [];
        foreach ($googleAdsHotelList->getAll() as $googleAdsHotel) {
            $adGroups[] = $this->adGroupGenerator-
>generate($googleAdsHotel, $defaultStatus, $campaignResourceName);
        }

        $this->adGroupRepository->setAccountId($customerId);
        $campaignId =
ResourceNameParser::parseToArray($campaignResourceName)['campaigns'];
        $splitGroupsResponse = $this->updateChecker-
>splitByUpdateOrCreate($adGroups, $customerId, $campaignId);
        $adGroups = array_merge($splitGroupsResponse-
>getEntitiesToCreate(), $splitGroupsResponse->getEntitiesToUpdate());

        $result = $this->adGroupRepository->save($adGroups);
```

```
        $createdAdGroups = $result->getResults();
        $errorMsg = $this->persistAdReference($createdAdGroups,
$googleAdsHotelList);

        $adGroupCreatorResponse = $this->errorToMessageParser-
>parsePartialFailureFromResponse($result, $errorMsg);
        return $adGroupCreatorResponse;
    }

    private function persistAdReference($createdAdGroups,
GoogleAdsEntityList $googleAdsHotelList): string
    {
        $repository = $this->adGroupReferenceRepository;
        $AdGroupRefsToSave = [];
        $errorMsg = '';
        /** @var MutateAdGroupResult $createdAdGroupResult */
        foreach ($createdAdGroups->getIterator() as
$createdAdGroupResult) {
            if ($createdAdGroupResult->getResourceName()) {
                try {
                    $resourceName = $createdAdGroupResult-
>getResourceName();
                    $createAdGroup = $createdAdGroupResult-
>getAdGroup();

                    if (!$createAdGroup) {
                        continue;
                    }

                    $listEntity = $googleAdsHotelList-
>getByAdGroupName($createAdGroup->getName());
                    $listEntity-
>setAdGroupResourceName($resourceName);
                    $listEntity->setAdGroupStatus($createAdGroup-
>getStatus());
                    $listEntity->setAdGroupId($createAdGroup-
>getId());

                    /* Persist the Adgroup-to-entity relationship in
database */
                    $adGroupRelationEntity = $this-
>adGroupReferenceRepository->find($resourceName);
                    if ($adGroupRelationEntity == null) {
                        //On Marketing's request, the baseline status
of References newly created during the AdGroup creation process will
be locked
                        $adGroupRelationEntity = new
AdGroupReference();
                        $adGroupRelationEntity->setLocked(true);
                    }
                    $adGroupRelationEntity-
>setIdentifier($googleAdsHotelList->getByAdGroupId($createAdGroup-
>getId())->getIdentifier());

                    list($customerId, $campaignId) = $this-
>parseCampaignResourceName($createAdGroup->getCampaign());
```

```
                    $adGroupRelationEntity-
>setAdGroupId($createAdGroup->getId());
                    $adGroupRelationEntity-
>setCampaignId($campaignId);
                    $adGroupRelationEntity-
>setResourceName($resourceName);
                    $adGroupRelationEntity-
>setAccountId($customerId);
                    $AdGroupRefsToSave[] = $adGroupRelationEntity;
                } catch (Throwable $e) {
                    $errorMsg .= "Found an error when attempting to
persist reference to an AdGroup that was created. Result received
that caused a failure: "
                        . $createdAdGroupResult-
>serializeToJsonString()
                        . "'. \nError message: "
                        . $e->getMessage() . "\n";
                }
            }
        }
        $repository->saveCollection($AdGroupRefsToSave);
        return $errorMsg;
    }
```

AdGroupAdCreator

Descripción

Genera y actualiza AdGroupAds/Ads/AdInfos en Google Ads

Métodos públicos

`public function create(GoogleAdsEntityList $googleAdsHotelList, $customerId, $campaignResourceName): CreatorResponse`

Este método funciona muy similar a su equivalente en AdGroup: Crea nuevas entidades adGroupAd, comprueba si se requieren creaciones o updates, y en caso afirmativo envía las creaciones y updates a Google Ads.

Sin embargo, debido a como se guardan los AdInfos, no se puede actualizar AdGroupAds cuyos Ads hayan cambiado directamente, hay que enviar los Ads modificados a través del repositorio de Ads.

KeywordCreator

Descripción

Crea nuevas keywords para los AdGroups recién creados. No se actualizarán las keywords de anuncios previamente creados (a petición de Marketing, ya que prefieren tener la opción de editar las keywords estándar a mano para afinar el targeting). Esto hace que sea mucho más sencillo que los otros dos Creators a nivel de lógica.

Métodos públicos

`public function create(GoogleAdsEntityList $googleAdsHotelList, $customerId): CreatorResponse`

Crea un set completo de keywords para cada GoogleAdsEntity de la lista que no tuviera un AdGroup previamente, y las envía a Google Ads.

Generadores

Servicios que generan entidades de Google individuales, listas para guardar con los repositorios.

AdGroupGenerator

Descripción

Genera elementos AdGroup, sin anuncios asociados, a partir de objetos GoogleAdsEntity.

Métodos públicos

```
public function generate(GoogleAdsEntity $googleAdsEntity, string $campaignResourceName, int $defaultStatus = null)
```

Genera un AdGroup sin anuncios ni entidades relacionadas, con todos los campos básicos, a partir de un GoogleAdsEntity. Permite pasar un status específico predeterminado para sobrescribir el status marcado en la GoogleAdsentity si se quiere.

AdGroupAdGenerator

Descripción

Genera AdGroupAds a partir de GoogleAdsEntity. Dado que un AdGroupAd no tiene sentido sin un Ad al que relacionar, los AdGroupAds siempre se devuelven con un Ad asociado.

Métodos públicos

```
public function generateFromAd(Ad $ad, GoogleAdsEntity $entity): AdGroupAd
```

En caso de que ya tengamos un Ad, este método permite crear un AdGroupAd y asociarle el Ad, así como el grupo al que apunte el GoogleAdsEntity

```
public function generateFromAdDefinitionSchema(GoogleAdsEntity $googleAdsEntity, $adDefinition): AdGroupAd
```

En caso de que NO tengamos un Ad, podemos utilizar este método para el sistema nos construya uno y lo asocie a un nuevo AdGroupAd relacionado con el GoogleAdsEntity, y lo devuelva.

Recibe como parámetro no solo el GoogleAdsEntity, sino también un schema adDefinition (para poder llamar a AdGenerator para construir el Ad).

AdGenerator

Descripcion

Genera entidades Ad completas con sus AdInfo. Incluye métodos privados para generar diferentes tipos de AdInfo, ya que un AdInfo no tiene sentido sin un Ad.

Métodos públicos

```
public function generateFromAdDefinitionSchema(GoogleAdsEntity $googleAdsEntity, array $adDefinition, array $existingAdGroupAdsForEntity): Ad
```

Genera un Ad completo, con su AdInfo correspondiente, basado en un schema json[5].

El método se encarga de llamar a los servicios necesarios para parsear el schema, reemplazar todos los comodines en el schema con los valores de la entidad GoogleAdsEntity, y preparar todos los enlaces antes de crear el AdInfo.

KeywordGenerator

Descripción

Genera entidades AdWordCriterion de tipo Keyword

Métodos públicos

`public function generate(GoogleAdsEntity $googleAdsEntity, $keywordText, $matchType): AdGroupCriterion`

Genera un AdGroupCriterion de tipo Criterion::KEYWORD, con el texto $keywordText y configurado para el tipo de coincidencia recibido en $matchType

Comparators

Comparan entidades de Google para comprobar si hay diferencias que requieran update.

Al ser las entidades de Google mensajes protobuf, es necesario hacer comparación por campos, pero el problema es que a menudo elementos del mismo tipo no tienen los mismos campos con el mismo nombre (esta inconsistencia de variables ha hecho complicado hacer algunos métodos genéricos, en general). Para hacer esto, he creado servicios Comparator.

Cada tipo de entidad principal tiene su propio Comparator, pero todos siguen el mismo esquema:

`public function isDifferent($newEntity, $alreadyUploadedEntity): bool`

Comprueba si las dos entidades son del mismo tipo, y si se han de considerar diferentes, según los campos que se haya decidido para cada entidad.

Un ejemplo del aspecto que puede tener un isDifferent sería como sigue:

```
public function isDifferent($newEntity, $alreadyUploadedEntity):
bool
    {
        //If there's no difference at all, no need to do indepth
search
        if (!parent::isDifferent($newEntity,
$alreadyUploadedEntity)) {
            return false;
        }

        /* Add special cases here */
        if ($newEntity->getAd()->getType() ===
AdType::RESPONSIVE_SEARCH_AD) {
                  //Responsive Ads require indepth search, ignoring
performance fields
            $exceptionFields = ['assetPerformanceLabel',
'policySummaryInfo'];
```

5 Ver Anexo 1 para ver un ejemplo del formato de un schema json.

```
            return parent::recursiveIsDifferent($newEntity,
$alreadyUploadedEntity, $exceptionFields);
        }
        /* ---- */

        return true;
    }
```

UpdateCheckers

Servicios que dividen listas de objetos de Google según si necesitarán enviarse como Create, Update, o no requieren modificaciones.

AdGroupUpdateChecker

Descripción

Permite saber si los AdGroups en una colección son nuevos, son diferentes a los que hay en Google Ads (y por tanto hay que hacer actualización), o son iguales a los que hay en Google (y por tanto no hay que enviar actualización).

Métodos públicos

`public function splitByUpdateOrCreate(array $objectsToSplit, $customerId, $campaignId): UpdateCheckerResponse`

Recibe un conjunto de elementos AdGroup, y devuelve una respuesta UpdateCheckerResponse con dos parámetros: toCreate (elementos que necesitan ser creados) y toUpdate (elementos que necesitan ser actualizados). Aquellos elementos en la lista que no necesiten ninguna de las dos cosas simplemente se eliminan de la lista.

AdGroupAdUpdateChecker

Descripción

Permite saber si los AdGroupAds (y sus Ads incluidos) en una colección son nuevos, son diferentes a los que hay en Google Ads (y por tanto hay que hacer actualización), o son iguales a los que hay en Google (y por tanto no hay que enviar actualización).

Métodos públicos

`public function splitByUpdateOrCreate(array $objectsToSplit, $customerId, $campaignId): UpdateCheckerResponse`

Recibe un conjunto de elementos AdGroupAd **que contengan Ads del mismo tipo**, y devuelve una respuesta UpdateCheckerResponse con dos parámetros: toCreate (elementos que necesitan ser creados) y toUpdate (elementos que necesitan ser actualizados). Aquellos elementos en la lista que no necesiten ninguna de las dos cosas simplemente se eliminan de la lista.

Para entender por qué necesitamos que los Ads sean todos del mismo tipo, veamos un fragmento de código del método performCheck que utiliza este servicio para comprobar si se requieren actualizaciones. Concretamente, la petición que utilizamos para obtener los AdGroupAds que ya existen.

```
$response = $this->adGroupAdRepository->findBy(
            $campaignId,
            [
                'ad_group_ad.ad.type' => AdType::name($sampleObject-
>getAd()->getType()),
                'ad_group.resource_name' => $adGroupNames
            ],
            $fields);
```

Si el lector recuerda el apartado sobre los repositorios, $fields son los campos que queremos buscar. El problema es que diferentes tipos de Ad tienen campos completamente diferentes en el AdInfo, tanto en nombre como en cantidad (por ejemplo: los Ads adaptables tienen un array de títulos, mientras que los Ads expandidos tienen tres columnas titulo1, titulo2, titulo3), e intentar hacer select a campos que no existen causa excepciones de API.

Así que las opciones habrían sido hacer una petición separada por cada AdGroupAd para poder personalizar la búsqueda, o, lo que he hecho, simplemente exigir como precondición que todos los Ads sean del mismo tipo (es decir, dividir por tipo de Ad antes de llamar al servicio), y de esta forma sabemos que todos los campos serán iguales, así que los obtenemos una vez y podemos obtener todas las líneas en un solo select. De esta forma, durante el flujo normal, en vez de hacer una petición a google por cada AdGroupAd, haremos una petición por cada tipo de AdGroupAd - de los cuales, ahora mismo, solo se han implementado dos.

Una vez los tenemos todos, simplemente filtramos:

```
foreach ($adGroupAdsToCheck as $adGroupAd) {
            if (array_key_exists($adGroupAd->getAdGroup(),
$foundAdGroupAds)) {
                $existingAd = $foundAdGroupAds[$adGroupAd-
>getAdGroup()];
                $adGroupAd->setResourceName($existingAd-
>getResourceName());
                $adGroupAd->getAd()->setResourceName($existingAd-
>getAd()->getResourceName());
                if ($this->comparator->isDifferent($adGroupAd,
$existingAd)) {
                    $toUpdate[] = $adGroupAd;
                }
            } else {
                $toCreate[] = $adGroupAd;
            }
        }
```

9.4 Timeline de la implementación

Al haber trabajado con SCRUM en la empresa, tengo la oportunidad de plasmar aquí una línea temporal aproximada de las tareas llevadas a cabo.

Al ser SCRUM, las tareas se organizan por la fecha en la que **comenzó** el sprint en el que se llevaron a cabo.

25/02/21 – Desglosar TFG Google Ads

11/03/21 – Estudiar cambio de librería de Ads

11/03/21 – EXTRA: Estudiar requisitos Ads y replantear

(como puede suponerse al intentar hacer pruebas con las librerías fue donde nos dimos cuenta del problema de que no podíamos poner esta integración en el servidor central, así que hubo que replantear)

25/03/21 – Montar nuevo docker y proyecto para Google Ads

08/04/21, 22/04/21 – implementar funcionalidades básicas Google Ads

(Esto incluía principalmente crear el session builder, los repositorios, y los generadores, y mucho tiempo dedicado a arreglar problemas inesperados)

22/04/21 – Montar caso de uso definitivo y command

20/05/21 – Modificar sistema de actualización de anuncios para minimizar updates innecesarios

(Nota explicativa: al crear el Command y discutirlo con mi supervisor, se decidió que había que replantear como se actualizaban los anuncios ya existentes, ya que cuando actualizamos un anuncio este se pausa durante unos diez minutos. Esto no es problema en el planteamiento inicial de actualizar cada 12 horas, pero si íbamos a poner el command cada hora, no podíamos perder 10 minutos de cada hora en tráfico. Aquí fue donde implementamos Comparators y UpdateCheckers)

A partir de aquí, ya teníamos un Command con funcionalidades básicas y la campaña existía y estaba funcionando, pero no había terminado de implementar lo que estaba proyectado aquí. Sin embargo, el ritmo se redujo bastante.

03/06/21 – Implementar reglas de negocio pausa o activación por porcentajes

08/07/21 – Modificar patrones de anuncios de Google Ads para añadir precios, cuentas atrás, días.

29/07/21 – Depurar logs de error de Google Ads de forma que sean comprensibles para usuario

12/08/21 – Subir versión Google Ads de V6 a V8

10.Metodología de testing

Debido a la naturaleza de este proyecto, que trabaja con APIs externas, desestimé hacer un sistema de tests unitarios automáticos para los casos de usos principales – primariamente porque el funcionamiento de Symfony es que **cuando se hace un deploy se pasan todos los tests unitarios creados para las clases**. Esto resultaría en que a cada deploy se hicieran todas las peticiones de los tests unitarios, y que usuarios haciendo cambios en el entorno de Google desde la interfaz externa pudieran romper las precondiciones de los tests.

De hecho, la persona que creó el sistema antiguo de AdWords sí que había creado un sistema de tests unitarios, y esto es parte del motivo por el que desestimé hacer lo mismo. Cuando llegué y abrí la campaña de test que se había creado, la campaña tenía aproximadamente 82400 AdGroups creados por el test unitario que había hecho para probar su caso de uso de crear AdGroup, ya que este se ejecutaba cada vez que se hacía un deploy en Esquiades. Y ni siquiera tenemos la opción de eliminar realmente anuncios, ya que aunque eliminemos un anuncio, Google lo mantiene en la tabla, simplemente lo marca con el status Eliminado y hace que no se muestre en la interfaz gráfica, pero sigue ralentizando las búsquedas. Esto hacía que cada select en la campaña de test rondara un minuto, y en ocasiones llegara a causar connection timeouts.

Además, parte del problema es que el entorno de prueba y el entorno real no utilizan las mismas políticas, y las políticas están sujetas a cambios externos, así que, para los casos de uso de creación de anuncios, tendríamos que estar cambiando los tests constantemente.

En vez de esto, la mayoría de mis tests han sido 100% manuales. Para facilitar las pruebas de los casos de uso que normalmente solo se llaman desde otros casos de uso, he creado un controlador TestController, que permite hacer diferentes acciones que prueban varios de los casos de uso básicos, siendo algo así como “pseudo tests unitarios”, o permiten al desarrollador comprobar precondiciones desde el navegador.

Estos métodos son simples acciones de controlador, que tienen este aspecto:

```
/**
* @Route("/test")
*/
public function responseTest()
{
   $testClient = $this->googleAdsSessionBuilder->build();
   return new Response('RESPONSE TEST');
}
```

Así, para casos de prueba, en local todo lo que tengo que hacer es acceder a https://google-ads.viajesparati.local/test desde el navegador, y se ejecuta la prueba.

Como entorno de test, he creado una nueva campaña de test para ubicar mis entidades de prueba (ya que la que existía previamente era inutilizable por razones anteriormente mencionadas).

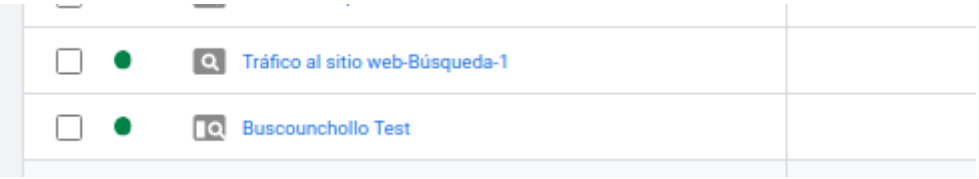

Ilustración 13: Campaña Test

10.1 Casos de prueba principales y como se han probado

Establecer conexión con Google

Para probar esto, que es básicamente el caso de uso A1, tengo un método sencillo que simplemente intenta ejecutar el constructor de sesión, sin un try/catch. En caso de que salte una excepción, Symfony hará aparecer el trace en el navegador. Si no hay ninguna excepción, simplemente muestra RESPONSE TEST (el código es el de ejemplo arriba).

Probar que las entidades funcionan

Como se ha mencionado, debido a la extraña construcción de las entidades protobuffer que usa google, Symfony hace saltar errores a partir de USER_WARNING al crear múltiples entidades.

Para comprobar fácilmente si esto está causando problemas, creé un método test que simplemente crea muchas entidades vacías en rápida sucesión. De esta forma si el sistema falla, comprobar si esto es la causa son dos clicks, incluso en producción.

Obtener y guardar entidades con repositorios

Para probar los findBy() y save() de los repositorios, tengo métodos que hacen queries de prueba prediseñadas, de forma que probar si los repositorios funcionan solo requiere entrar a las direcciones /adGroupTest, /adGroupSaveTest, /adTest, etc.

Presento aquí un ejemplo de método de test de findBy en AdGroup (encuentra todos los AdGroups que están pausados o activos) ya que son todos prácticamente idénticos, simplemente cambiando las entidades.

```
/**
     * @Route("/adGroupTest")
     */
    public function adGroupTest()
    {
        $accountId = $this->googleAdsAccountsGetter-
>getFromTag('buc');
        $campaignId = 1358099459;

        $this->adGroupRepository->setAccountId($accountId);
        $rows = $this->adGroupRepository->findBy($campaignId,
['ad_group.status' => ['ENABLED', 'PAUSED']]);
        $responseArray = [];
        foreach ($rows as $googleAdsRow) {
            /** @var GoogleAdsRow $googleAdsRow */
```

```
            $responseArray[] = $googleAdsRow-
>serializeToJsonString();
        }

        return new Response(json_encode($responseArray));
    }
```

Y el resultado, en el navegador, tiene este aspecto:

Ilustración 14: Pantalla de resultados de un GET

Detectar si entidades necesitan update

Los updateCheckers, realmente, no hacen lógica compleja. El *punto de fallo* son las comparaciones que detectan si se considera que el elemento necesita ser actualizado.

Así pues, el caso que realmente he testeado con cuidado para probar los UpdateCheckers es comprobar que los comparadores funcionan. Para cada comparador, tengo un método /testCompare{tipo de entidad} más o menos con este aspecto:

```
/**
     * @Route("/testCompareAdGroups")
     */
    public function testCompareAdGroups(GenericAdsEntityComparator
$comparator)
    {
        $original = new Ad();
        $original->setName('AdGroup 1');
        $original-
>setType(AdGroupTypeEnum\AdGroupType::DISPLAY_STANDARD);
        $original->setExpandedTextAd(new ExpandedTextAdInfo([
            'headline_part1' => 'Test1' ,
            'headline_part2' => 'Test2',
            'description' => 'Test3'
        ]));

        $modified = new Ad();
        $modified->mergeFrom($original);
        $modified->getExpandedTextAd()->setHeadlinePart1('Change
test');
        $modified->setDisplayUrl('https://test.com');

        $testAd = json_decode($original->serializeToJsonString(),
true);
        $test1 = $comparator->isDifferent($original, $modified);
```

```
        $test2 = $comparator->isDifferent($original, $original);

        return new JsonResponse([$test1, $test2]);
    }
```

Generar nueva entidad

Similarmente, para probar los generadores, tengo los siguientes métodos: /testBuildAdGroup, /testBuildAd (que realmente hace AdGroupAds + Ad, porque no puede existir un Ad sin ningun AdGroupAd), /testBuildKeyword.

Todos los métodos siguen el mismo sistema: generamos una GoogleAdsEntity, la utilizamos con el generador que toque, y la guardamos con un repositorio.

Como ejemplo, para ver la estructura, veamos /testBuildAd:

```
/**
     * @Route("/testBuildAd")
     * @param AdGenerator $adGenerator
     * @param AdGroupAdRepository $adGroupAdRepository
     * @param GoogleAdsAccountsGetter $accountsGetter
     * @return Response
     */
    public function testBuildAd(AdGenerator $adGenerator,
AdGroupAdRepository $adGroupAdRepository, GoogleAdsAccountsGetter
$accountsGetter)
    {
        $testEntity = new GoogleAdsEntity();
        $testEntity->setIdentifier('Test Identifier');
        $testEntity->setName('This is an extremely long hotel name
for testing');
        $testEntity->setAdGroupName('This is an adgroup name');
        $testEntity->setUrl('https://www.testurl.com');
        $adDefinition =
AdInfoSchemas::getAdDefinitionByTag('buc_responsive_search');

        $customerId = $accountsGetter->getFromTag('buc');
        $adGroupId = '123858997644';

        $testAd = $adGenerator-
>generateFromAdDefinitionSchema($testEntity, $adDefinition, []);
        $adGroupAd = new AdGroupAd([
            'ad_group' => ResourceNames::forAdGroup($customerId,
$adGroupId),
            'ad' => $testAd
        ]);

        $adGroupAdRepository->setAccountId($customerId);
        $validateResult = $adGroupAdRepository->save([$adGroupAd],
['validateOnly' => true]);

        return new Response($validateResult-
>serializeToJsonString());
    }
```

Crear anuncios completos

La prueba real de esto es simplemente ejecutar el command sobre la campaña de test.

Precondiciones especiales:

En entorno local, tener las bases de datos de Buscounchollo, Esquiades, y GoogleAdsProject

Resultados esperados:

Nuevos anuncios completos creados en campaña buscounchollo test, entidades creadas en AdGroupReference, log sin errores.

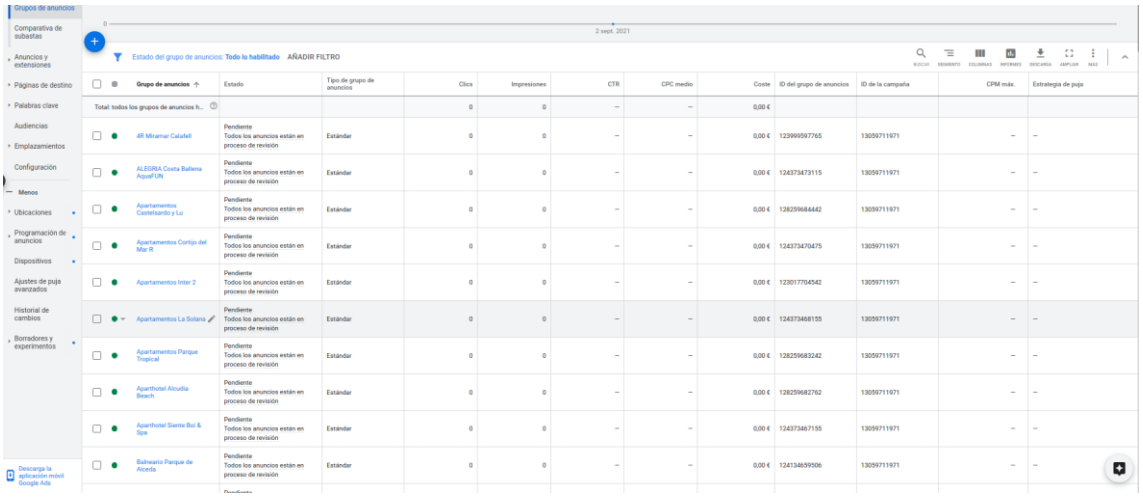

Ilustración 15: AdGroups creados

WHERE ORDER BY

	identifier	ad_group_id	campaign_id	resource_name	account_id	locked
1	642	123017703262	13059711971	customers/2015311054/adGroups/123017703262	2015311054	1
2	87598	123017703302	13059711971	customers/2015311054/adGroups/123017703302	2015311054	1
3	16831	123017703342	13059711971	customers/2015311054/adGroups/123017703342	2015311054	1
4	19816	123017703502	13059711971	customers/2015311054/adGroups/123017703502	2015311054	1
5	1584	123017703542	13059711971	customers/2015311054/adGroups/123017703542	2015311054	1
6	18469	123017703582	13059711971	customers/2015311054/adGroups/123017703582	2015311054	1
7	7341	123017703742	13059711971	customers/2015311054/adGroups/123017703742	2015311054	1
8	363	123017703782	13059711971	customers/2015311054/adGroups/123017703782	2015311054	1
9	601	123017703822	13059711971	customers/2015311054/adGroups/123017703822	2015311054	1
10	40928	123017703982	13059711971	customers/2015311054/adGroups/123017703982	2015311054	1
11	1612	123017704022	13059711971	customers/2015311054/adGroups/123017704022	2015311054	1
12	20	123017704062	13059711971	customers/2015311054/adGroups/123017704062	2015311054	1
13	220	123017704222	13059711971	customers/2015311054/adGroups/123017704222	2015311054	1
14	83891	123017704262	13059711971	customers/2015311054/adGroups/123017704262	2015311054	1
15	14923	123017704302	13059711971	customers/2015311054/adGroups/123017704302	2015311054	1
16	3367	123017704462	13059711971	customers/2015311054/adGroups/123017704462	2015311054	1
17	68334	123017704502	13059711971	customers/2015311054/adGroups/123017704502	2015311054	1
18	158	123017704542	13059711971	customers/2015311054/adGroups/123017704542	2015311054	1
19	81599	123017704702	13059711971	customers/2015311054/adGroups/123017704702	2015311054	1
20	99002	123017704742	13059711971	customers/2015311054/adGroups/123017704742	2015311054	1
21	815	123017704782	13059711971	customers/2015311054/adGroups/123017704782	2015311054	1

Ilustración 16: Referencias creadas

11. Resultados tras la puesta en marcha

A fecha de escribir este apartado, este proyecto lleva funcionando ya más de un mes en producción en la empresa. De hecho, todas las ilustraciones en el apartado 3.2 Estructura conceptual de Google Ads son de elementos creados por mi proyecto. Así que creo apropiado dedicar aunque sea media página a mostrar una pincelada de los resultados obtenidos en entorno real.

Si bien es cierto que se está refinando, y estamos pensando mejoras, el funcionamiento básico ya ha dado un resultado espectacular. A día de hoy, la campaña mantenida por mi proyecto es la segunda campaña de búsqueda con más dinero invertido entre todas las de la empresa, el presupuesto diario para la campaña es de media 3.000 euros, y el dinero total invertido ronda los 130.000 euros.

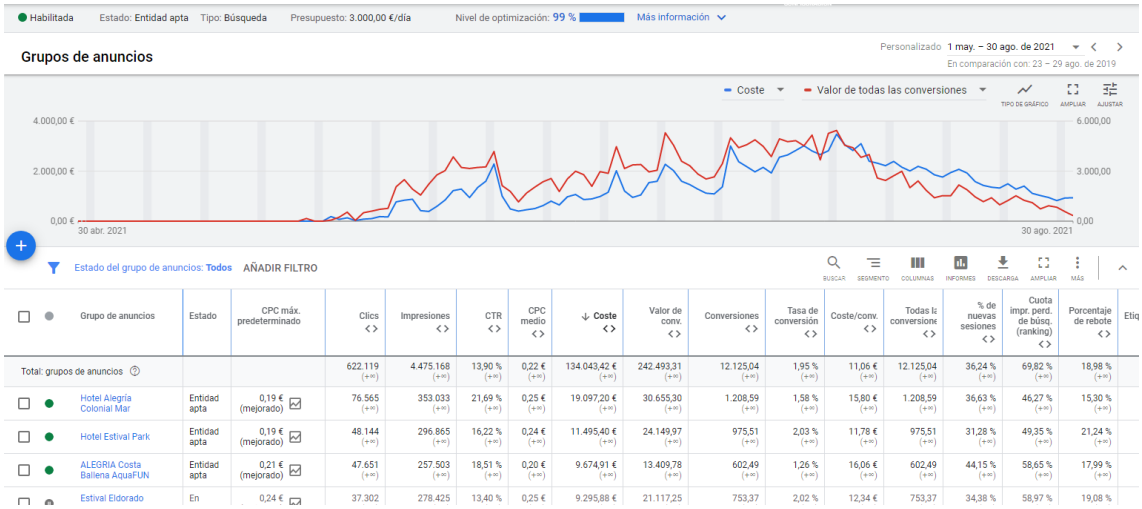

Ilustración 17: Métricas de la campaña

El CTR (clickthrough rate, básicamente el ratio de clientes que al ver el anuncio hacen click) de la campaña es sorprendentemente elevado, con algunos de los mejores anuncios llegando a CTR por encima del 20% (como perspectiva, un CTR por encima del 10% se considera muy bueno) y una tasa de conversión (el porcentaje de personas que, después de hacer click, inmediatamente compran) mediando alrededor de un 2%. Y ha de tenerse en cuenta que la tasa de conversión en sitios de viajes siempre es artificialmente baja, ya que el flujo normal de un cliente buscando un viaje es entrar al sitio la primera vez mediante anuncio, pensárselo durante unos días, y volver al sitio buscando el sitio de forma directa cuando se ha tomado la decisión para realizar la compra. Y en este caso, al no ser la visita en la que se viene desde el anuncio la misma en la que se hace la compra, este cliente no cuenta para la tasa de conversión.

Según marketing, una gran parte del motivo de estos resultados tan espectaculares es que, al automatizar anuncios a partir de patrones, podemos servir no solo los nombres genéricos del chollo, sino nombres de hoteles específicos y otros elementos similares, y al estar automatizado podemos servir todos, no solo la subsección de los más populares (en la cual, lógicamente, la competencia por las pujas de anuncios sería más feroz).

Ya se está hablando de aprovechar la extensibilidad del proyecto para empezar a preparar una campaña para Esquiades.com con vistas al invierno.

12. Conclusiones

12.1 Aprendizaje

Durante el transcurso de este proyecto he tratado casi por entero con elementos que no había utilizado nunca, así que ciertamente he tenido oportunidades de aprender una gran cantidad de cosas.

Una breve lista de temas que he aprendido prácticamente desde cero:

- Docker: He tenido oportunidad de aprender cómo funciona Docker y la creación de un docker nuevo, la sintaxis de un Dockerfile, como se configura un Docker compose, como hacer troubleshooting cuando no funciona… antes de empezar este proyecto, ni siquiera sabía *qué* era un Docker. Ahora, si bien no me atrevo a decir que pudiera configurar uno desde cero (ya que una parte importante de la puesta en marcha estaba cubierta por la infraestructura de la empresa), me atrevería a decir que podría hacer troubleshooting a una instalación de Docker existente.
- Servicios Cloud: Para poner en marcha el proyecto y subirlo a Amazon ECS, tuve por primera vez la oportunidad de ver el funcionamiento de un sistema de contenedores en nube.
- Frameworks: Si bien ya he estado trabajando en entornos Symfony en la empresa desde hace un tiempo, hay una diferencia importante entre "trabajar en un sistema existente que utiliza un framework" y literalmente "crear un proyecto nuevo desde cero utilizando el framework". He podido aprender cómo se crean los archivos de configuración, como montar un composer, un autoload, como utilizar los mapeos de Symfony, como funciona todo el sistema de debugging y excepciones (debido a una serie de problemas con las entidades de Google) a bajo nivel, como configurar un fichero htaccess para que Symfony utilice Apache correctamente… Incluso probé a aprender cómo funcionan las migraciones, para la base de datos, pero después de aprender cómo funcionaban e intentar usarlas, desestimé su uso.
- Sistemas de Google: Incluso antes de poder entrar a Google Ads en sí, aprender a tratar con Google es un verdadero mundo. Las documentaciones de Google son completas (para los elementos que realmente son importantes para ellos, al menos), pero a menudo impenetrables y poco accesibles. De hecho, perdimos varias semanas de trabajo porque empecé a trabajar con la librería de AdWords en vez de Ads, ya que los nombres eran extremadamente similares. Los mecanismos de las cuentas de Google y la autenticación no son simples, las librerías varían enormemente…

  En general Google es prácticamente un lenguaje diferente, que tienes que aprender si pretendes operar con ellos. Este proyecto ha sido una oportunidad de oro para aprender este "lenguaje de Google", que me servirá para muchas más cosas que solo para hacer Ads.
- Google Ads en sí: Sin duda la parte de la que más he aprendido. Cuando empecé este proyecto, esperaba simplemente que Google Ads fuese un endpoint REST corriente para subir anuncios, pero resultó ser mucho más. Ads es un sistema extremadamente complejo, basado en una API que supera las 700 páginas. Este sistema que yo he montado solo rasca la superficie – los AdGroups tienen acceso a métricas, y podemos crear políticas mediante la API, incluso podemos asignar

diferentes estrategias de puja, crear feeds… las opciones son infinitas. Y con este proyecto, si bien no he tocado todos los puntos de la API, ciertamente he aprendido como utilizarla y como se organizan sus entidades y servicios, de forma que a partir de aquí crear nuevas funcionalidades será sencillo. Tras completar este proyecto, me veo perfectamente capacitado para crear funcionalidades más complejas con Ads.

- Marketing: crear este proyecto me ha dado una excusa para poder relacionarme con el equipo de marketing y aprender realmente como funciona el mundo de los anuncios online, la terminología de marketing, como evaluar ratios de clickthrough, como poder analizar si un anuncio es bueno o malo… en general toda una serie de cosas que, si bien no son de utilidad directa en una carrera de informática, son ciertamente interesantes y que me alegro de haber aprendido.

12.2 Elementos a mejorar en el proyecto

Si he de ser honesto, infinitos. Conforme creaba el proyecto, yo y mis tutores de empresa íbamos teniendo ideas, pero era suficientemente consciente del riesgo de scope creep para no implementarlas todas (si bien es cierto que este documento, a día de entrega, ya no es exhaustivo, porque ya se han añadido elementos que no están en esta documentación, tal es la claridad con la que veíamos posibles mejoras).

Una simple lista de algunos elementos que se quedaron en el tintero:

- Sistema para analizar las métricas en la interfaz de la empresa, comparando resultados en Google con facturación real total: como he comentado en el apartado de resultados, un problema que tenemos es que la tasa de conversión de Google es completamente irreal, debido a como funciona el mercado de viajes. Una idea que tenemos es crear una vista que permita comparar gráficamente la facturación de un chollo con su STR y tasa de conversión en Google.
- Sistema de selección de schemas para los Ads basados en características del chollo: Esto ya ha sido parcialmente implementado, con un schema particular para chollos de categoría Top, pero la idea es poder tener una docena de schemas prediseñados y elegir el más apropiado.
- Reglas inteligentes para estimar un CPC de puja: Esto, de momento, se ha desestimado, principalmente por el nivel de coste en horas implicado. Crear un sistema que pueda analizar y con suficiente detalle para estimar una puja recomendada sería extremadamente complejo, así que de momento esto se dejará para más adelante.
- Sistema que recupere metrics de cada anuncio para detectar anuncios con poco éxito y reducir CPC para reducir costes: los costes malgastados ya se han reducido tremendamente con este sistema, pero podemos ajustarlos todavía más.
- Subida de versión: Cuando empecé el proyecto, la última versión de la API era V6. Para cuando lo acabé, estábamos en V8 y Google pretende deprecar V6 en breves. Así que esto es bastante prioritario. Las nuevas versiones también incluyen control de errores afinado para los Ads, que en la versión V6 todavía no tenían la capacidad de fallidas parciales que usamos para hacer los logs de los AdGroups y las Keywords. (Implementado recientemente)

12.3 Valoración personal

A decir verdad, este proyecto ha sido extremadamente interesante, y el tipo de reto que no me importaría volver a afrontar.

Por un lado, es cierto que ha dado una gran cantidad de quebraderos de cabeza. Sobre todo para su implementación inicial (en ningún momento había esperado que intentar utilizar una API REST acabaría resultando en tener que aprender como montar un Docker), pero no solo eso. Durante este proyecto he encontrado una enorme cantidad de sorpresas y problemas que he tenido que afrontar, desde los más sencillos, a extraños fallos de configuración que me obligaron a estudiarme la documentación de XDebug o el sistema de versionado de pecl. Siendo la muestra más clara el problema que tuve con las entidades protobuffer de Google – más de una semana experimentando con versiones de librerias protobuf, grpc, incluso php y debug, sin entender el problema, hasta descubrir que el problema era simplemente que las entidades protobuf se definen de una forma que causa repeticiones, y Symfony promueve el warning que esto causa a un full Error. Y por supuesto, siempre está el problema de encontrar tiempo para hacer el trabajo sin descuidar mis otras responsabilidades en la empresa.

Pero, por otro lado, lo que he aprendido realmente no lo hubiera podido aprender de otra manera. Y puedo decir que aprender a utilizar las APIs de Google ya me está sirviendo – cuando escribo estas conclusiones, la empresa me ha asignado a implementar el sistema de llamadas verificadas para la aplicación Phone de Google en los sistemas de la empresa, de forma que cuando llamemos a clientes pueda aparecer nuestro logo y membrete. Algo que sería mucho mas duro de implementar si no fuese porque gracias a este proyecto ya tengo práctica con las APIs de Google.

En general, estoy bastante satisfecho con los resultados – aunque es cierto que me hubiera gustado poder hacer un poco más en algunos puntos, y los cambios de dirección a mitad de proceso han resultado en algunas partes del código no siendo todo lo claras que me gustaría

Me gustaría dedicar también un momento a agradecer al equipo de sysadmins de Viajes Para Ti su ayuda con todo el proceso de puesta en marcha. Dudo mucho que hubiese podido acabar este proyecto a tiempo sin su ayuda.

13. Recursos utilizados

[1] Documentación Google Ads API

https://developers.google.com/google-ads/api/docs/release-notes

[2] Documentación Symfony,

https://symfony.com/doc/current/index.html

[3] Documentación Docker

https://docs.docker.com/

[4] Symfonycasts tutorials

https://symfonycasts.com/tracks/symfony

[5] Foro devs Google

https://groups.google.com/g/adwords-api

Anexo 1: Schema de ejemplo

```
BUC_SEARCH_AD_SCHEMA_COUNTDOWN = [
        'adInfo' => 'responsive_search_ad',
        'type' => AdType::RESPONSIVE_SEARCH_AD,
        'class_info_name' => ResponsiveSearchAdInfo::class,
        'schema' => [
            'headlines' => [
                '##NAME##',
                'Reserva Al Mejor Precio',
                '{KeyWord:##ADGROUPNAME##}',
                'Oferta ##NAME##',
                'Consigue Ahora Tu Chollazo',
                'A la Venta por Tiempo Limitado',
                'Tu Escapada A Precio De Chollo',
                'Múltiples Fechas Disponibles',
                'Con Opiniones de Clientes',
                'Excelente Atención al Cliente',
                'Te Mereces Unas *Vacaciones*',
                'No Te Quedes Sin Tu Chollo',
                '##NIGHTS_TEXT## ##PRICE##€',
                'El 98% de Clientes Satisfechos',
                'Reserva Antes De Que Se Agote'

            ],
            'descriptions' => [
                '##NAME## ##NIGHTS_TEXT## ##PRICE##€, ¡Entra y
Escápate!',
                'Ofertón Para ##NAME## por tiempo limitado.
¡Reserva!',
                'Si Te Interesa Este Chollo, Resérvalo Antes De Que Se
Agote. ¡Quedan {=GLOBAL_COUNTDOWN("##END_DATE##","es",0)}!',
                '¿Buscas El ##NAME##? Encuéntralo Aquí a Un Precio
Exclusivo.',
            ],
            'path1' => 'hotel',
            'path2' => '##NAME##'
        ]
    ];
```