UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Programa de Apoyo al Egreso de Profesionales en Actividad

TÍTULO: Abstracción del motor de búsqueda utilizado en una aplicación web legacy a partir de "Search Service" un Middleware de búsqueda optimizado

AUTOR: Gauna Lucas Emiliano

DIRECTOR ACADÉMICO: De Giusti Armando

DIRECTOR PROFESIONAL: Querejeta Ramiro

CARRERA: Licenciatura en Sistemas

Resumen

En esta Tesina de grado se describe el trabajo de desarrollo realizado para reemplazar el servicio de búsqueda utilizado en una aplicación web legacy. Con el nuevo servicio de búsqueda se busca agregar una capa de abstracción con el fin de mejorar las salidas a producción. El análisis incluye la descripción de los modelos de datos creados en función de los distintos tipos de respuestas que se pueden obtener del nuevo servicio, el detalle de las configuraciones requeridas para su funcionamiento, el procesamiento de resultados así como también un trabajo de analítica de datos para recolectar información útil sobre los términos más buscados y seleccionados por los usuarios.

Palabras Clave

Abstracción, analítica de datos, motor de búsqueda, servicio, aplicación web, flujo de ejecución, migración

Conclusiones

Con la integración del servicio se logró la capa de abstracción deseada. Se eliminaron las salidas a producción de la aplicación web ante cambios en el servicio de búsqueda y se redujeron los costos que esto implicaba. Con el trabajo de analítica de datos, se logró identificar cuales son los términos más buscados por los usuarios lo que permitió ajustar la relevancia de los resultados e incrementar la eficacia de la búsqueda.

Trabajos Realizados

Se creó un nuevo flujo de ejecución dentro de la aplicación web para integrar a "Search Service" como servicio principal de búsqueda. Una vez validado su funcionamiento se removió la lógica relacionada a Elasticsearch para optimizar el código. Se realizó un trabajo de analítica de datos, el cual permitió recopilar información de los términos más buscados por los usuarios.

Trabajos Futuros

Dado que la aplicación web legacy es realmente vieja, como trabajo futuro se plantea la migración de las páginas de búsqueda a una aplicación web construida a partir de tecnologías nuevas, tales como: Angular (versión 14) y Node.js con el objetivo de seguir desacoplando la aplicación web legacy hasta que finalmente quede obsoleta.

Fecha de la presentación: Diciembre 2022

Índice general

1. Introducción
   - 1.1. Motivación
   - 1.2. Objetivo y resultados esperados
2. Search Service
   - 2.1. Qué es
   - 2.2. Endpoints
     - 2.2.1. Multi-search
     - 2.2.2. Single-search
     - 2.2.3. Respuesta de Search Service
3. Arquitectura de la aplicación
4. Comienzo de la migración
   - 4.1. Análisis del código legacy
   - 4.2. Search Service Toggle
   - 4.3. Implementación y flujo de ejecución
     - 4.3.1. Search Controller
     - 4.3.2. Search Service Workflow
     - 4.3.3. Search Service Command
5. Modelo de datos
   - 5.1. Mappers
   - 5.2. Procesamiento de los resultados de búsqueda
6. Listado principal de búsqueda
7. Barra de navegación y barra secundaria
   - 7.1. Implementación
     - 7.1.1. Action Controller
     - 7.1.2. Search Service Workflow
     - 7.1.3. Single Search Command
     - 7.1.4. Manejo de errores
     - 7.1.5. Verificación de funcionamiento
8. Analitica de datos
   - 8.1. Página y lugar desde donde se disparó la búsqueda
   - 8.2. Eficacia del servicio de búsqueda
   - 8.3. Endpoint de Selección
   - 8.4. Implementación
   - 8.5. Gráficos y reportes
     - 8.5.1. Gráficos relacionados al servicio en general
     - 8.5.2. Gráficos relacionados a cada uno de los parques temáticos
9. Conclusiones y trabajos futuros

Glosario

1. Plugin: Un plugin es un script de Javascript, un fragmento de código el cual se ejecuta en el navegador, y sirve para añadir funcionalidad y dinamismo a un sitio web.
2. Middleware: En el contexto de este documento, es una aplicación que funciona como una capa de abstracción entre la aplicación web y el servicio real de búsqueda.
3. MVC: hace referencia al estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario y la lógica de control.
4. Rollback: En el contexto de una salida a producción de una aplicación web, hace referencia a la operación que devuelve a dicha aplicación a una versión o estado anterior. Generalmente se hace para volver a un estado de consistencia en donde todo funciona como es debido.
5. Hotfix: En el contexto de una salida a producción, un hotfix es lo que se conoce como un parche de código el cual requiere de una nueva salida a producción y cuya finalidad es solucionar un problema de gravedad.
6. UI: es el acrónimo que se va utilizar a lo largo del documento para hacer referencia a la interfaz de usuario.
7. Elasticsearch: Elasticsearch es un motor de búsqueda y analítica distribuido, gratuito y abierto para todos los tipos de datos, incluidos textuales, numéricos, geoespaciales, estructurados y no estructurados. Provee una interfaz web RESTful y con documentos JSON.
8. REST: hace referencia al estilo de arquitectura de software que se utiliza para realizar comunicaciones cliente/servidor. Se basa en el protocolo HTTP, y la respuesta de este tipo de servicios suelen ser en formato JSON o XML.
9. Frontend: Es la parte del desarrollo web que se dedica a la parte frontal de una aplicación web, en otras palabras es todo lo que está relacionado directamente con la interfaz de usuario y la comunicación con los servicios.
10. JSON: Es un formato de texto sencillo para el intercambio de datos. Se trata de un subconjunto de la notación literal de objetos de JavaScript, aunque, debido a su amplia adopción como alternativa a XML, se considera un formato independiente del lenguaje.
11. Endpoint: son las URLs de un API o un servicio backend que responden a una petición con la información solicitada.
12. Postman: Es una aplicación que nos permite realizar pruebas a una API. Es un cliente HTTP que nos da la posibilidad de testear servicios a través de una interfaz gráfica de usuario.
13. API: El término API es una abreviatura de Application Programming Interfaces, que en español significa interfaz de programación de aplicaciones
14. GIT: Es un sistema de control de versiones distribuido, el propósito principal de este sistema es el de llevar un control de todos los cambios realizados en una aplicación.

Capítulo 1

Introducción

El trabajo propuesto en esta Tesina está ligado a una aplicación web perteneciente a la empresa de entretenimientos más grande del mundo. Dicha empresa se encarga del manejo de distintos parques temáticos, así como también de los resorts vacacionales y todo tipo de productos comerciales. La misma cuenta con una aplicación web la cual brinda información relevante al usuario sobre todas las actividades que se pueden realizar dentro de los distintos parques.

Es debido a esta gran variedad de actividades que dicha aplicación cuenta con una página principal de búsquedas, la cual le permite al usuario ingresar un término en el buscador y obtener así resultados de todo tipo: atracciones, entretenimientos, spas, restaurantes, tiendas, destinos, eventos, etc. Los resultados varían según el parque sobre el cual se realiza la búsqueda. Cada uno de los parques temáticos cuenta con su propia página de búsqueda, las cuales se encuentran bajo dominios web diferentes.

Además de la página principal de búsqueda, la aplicación cuenta con páginas de listado donde el usuario puede ver un listado de resultados pertenecientes a un mismo grupo. Con “grupo” nos referimos a las categorías mencionadas anteriormente, es decir que hay páginas de listado para las atracciones, para los entretenimientos, y así sucesivamente. Cada una de estas páginas de listado cuenta a su vez con un input de tipo texto, que es inyectado por un plugin[1] de jQuery, el cual le permite al usuario obtener resultados acotados al listado en el que se encuentra. Este input cuenta con la característica de ser dinámico, es decir que le permite ver al usuario resultados a medida que va tipeando el término deseado. Esta funcionalidad es conocida internamente como “Busca mientras escribes”.

Este trabajo está enfocado en la migración del servicio de búsqueda que se utiliza en la aplicación web. Esta migración impacta tanto la página principal de búsqueda, como las páginas de listado, y todos los lugares de la aplicación desde donde se puede disparar una búsqueda. A grandes rasgos, el análisis incluye la descripción de los modelos de datos creados en función de los distintos tipos de respuesta que se pueden obtener del nuevo servicio, el detalle de cómo fue el proceso de desarrollo y el procesamiento de los resultados, así como también un trabajo de analítica de datos que sirve para recolectar información útil sobre los términos más buscados y seleccionados por los usuarios. Esto es para ambas funcionalidades: la página principal de búsqueda y “Busca mientras escribes”.

A continuación se presenta la motivación de esta tesina, así como el objetivo de la misma, y los resultados esperados.

1.1 Motivación:

La aplicación web utiliza Elasticsearch[7] como motor de búsqueda. Además de proveer de un motor de búsqueda de texto completo, Elasticsearch también provee una interfaz REST[8] la cual se consume desde la aplicación web para obtener los resultados. Toda la lógica necesaria para consumir la API[13] de Elasticsearch se encuentra dentro del código base de la aplicación.

El problema que presenta la implementación actual, es que la aplicación no se puede abstraer del motor de búsqueda utilizado, por lo que ante cualquier cambio que se tenga que hacer, por mínimo que sea, es necesario realizar una salida a producción. Todo lo que ese despliegue conlleva puede llegar a ser muy costoso: se requiere de tiempo, de recursos, y de coordinar a muchos equipos para poder garantizar una salida a producción limpia y sin problemas. Esta fue la principal razón por la que se decidió abstraer el motor de búsqueda y crear un nuevo servicio llamado “Search Service” que funcione como middleware[3] entre la aplicación web y la API de Elasticsearch.

1.2 Objetivo y Resultados esperados:

El principal objetivo de este trabajo es exponer el análisis y el trabajo de desarrollo realizado para abstraer el motor de búsqueda utilizado y reemplazarlo con “Search Service”, un middleware de búsqueda optimizado. Cabe mencionar que este trabajo no consiste en la creación del nuevo servicio de búsqueda, sino que está enfocado en todo el trabajo de frontend[10] necesario para adaptar la aplicación web y realizar así, todas las peticiones de búsqueda a través de Search Service.

Con esta migración se espera que cualquier actualización que se requiera del lado de Elasticsearch, ya sea cambiar una configuración, o algo más complejo como actualizar el motor de búsqueda a una nueva versión, sea posible y transparente para la aplicación web. Ya no serán necesarias las salidas a producción, por lo que estaremos ahorrando en recursos, en tiempo y por supuesto en dinero.

Además de la ventaja mencionada anteriormente y desde un punto de vista más tecnológico, este nuevo servicio nos provee de una API más sencilla, y de respuestas en formato JSON[11] más livianas y con estructuras mejor organizadas, lo cual mejora el tiempo de respuesta y simplifica toda la lógica y el procesamiento requerido para crear los distintos modelos de datos y mostrar los resultados en la UI[7].

En cuanto al trabajo de analítica de datos mencionado, se espera poder recolectar información sobre los términos más buscados en el sitio, los resultados de búsqueda más seleccionados por los usuarios así como también información sobre la página y el lugar desde donde se realizó la búsqueda. Toda la información recolectada se usará para la construcción de gráficos y posible toma de decisiones.

Capítulo 2

Search Service

2.1 Qué es

Search Service es una aplicación de tipo Web Service desarrollada en Java, la cual funciona como middleware entre las aplicaciones web y mobile. Provee de una API mucho más simple que la que provee Elasticsearch, así como también de características más avanzadas como por ejemplo: ajuste de relevancia de resultados, autenticación, analytics, y traducción de los resultados en diferentes idiomas. También protege a aquellas aplicaciones que lo utilizan ante los cambios generados en Elasticsearch luego de actualizar la versión de dicho motor de búsqueda.

2.2 Endpoints

Para la migración del servicio de búsqueda vamos a utilizar 3 de los endpoints[11] que provee Search Service para ser consumidos por la aplicación web. El primero de ellos, y el que va a alimentar a la página principal de búsqueda es el endpoint conocido como "*multi-search"* el cual retorna resultados de todo tipo: spas, eventos, restaurantes, etc. Si no se especifican las entidades sobre las cuales se va a realizar la búsqueda, este endpoint es el equivalente a realizar un búsqueda global sobre todas las entidades disponibles en el sitio. Luego vamos a estar utilizando el endpoint "*single-search"* para las búsquedas de tipo "Busca mientras escribes" en las páginas de listado. A diferencia del endpoint "*multi-search"* este retorna resultados para una entidad en particular. Y por último, vamos a utilizar un endpoint específico para el trabajo de analytics. Por una cuestión organizativa el endpoint utilizado para el trabajo de Analytics será descrito en el apartado correspondiente.
Cada uno de estos endpoints acepta POST como método HTTP, y retornan respuestas en formato JSON las cuales van a ser procesadas en la UI.

Todas las peticiones realizadas a los endpoints de búsqueda se realizan sobre un parque o "brand" en particular. Recordemos que la aplicación web sirve a varias páginas de búsqueda, una para cada parque.

2.2.1 Multi-Search

Este endpoint se utiliza para realizar búsquedas sobre múltiples entidades, y retorna una lista de resultados por cada una de las entidades especificadas en la petición, así como también una lista de resultados sin filtrar.

A continuación se detallan algunos de los parámetros aceptados por este endpoint. No se incluyen todos los parámetros que Search Service acepta ya que no son relevantes para el trabajo en cuestión.

Nombre	Requerido	Descripción
brand	si	Parque sobre el cual se realiza la búsqueda
q	si	Término de búsqueda. Tiene que estar codificado en caso de incluir caracteres especiales.
entityTypes	no	Lista de entidades delimitadas por "||" sobre las cuales se quieren obtener resultados. Search Service retorna una lista de resultados por cada entidad en el orden en el que fueron recibidas. Así como una lista de resultados sin filtrar lo cual equivale a una búsqueda global.
from	no	Utilizado para la paginación de resultados. Retorna una lista a partir del índice recibido.
size	no	Número máximo de resultados deseados. El default es de 20 registros. Se utiliza en conjunto con el parámetro "from".
type	no	El tipo de búsqueda que se desea realizar ya sea tradicional o "Busca mientras escribes". El default es la búsqueda tradicional.
entityType	no	Solo se usa para búsquedas de tipo single-search. A diferencia del parámetro "entityTypes" este solo admite un valor.
pageName	no	Identificador de analytics el cual determina desde qué página o aplicación la búsqueda fue iniciada.
pageEvent	no	Identificador de analytics el cual determina desde qué lugar de la página se disparó la búsqueda. Por ejemplo: desde la barra de búsqueda global, la barra de búsqueda secundaria, desde el listado principal de búsqueda, etc.

Tabla 1: Lista de algunos parámetros admitidos por Search Service

```
/multi-search?brand=parkId&entityTypes=restaurant||attraction&q=discount
```

Ejemplo de cómo luce el endpoint multi-search. Se excluye el dominio del servicio ya que no es relevante.

2.2.2 Single-Search

A diferencia del endpoint anterior, single-search realiza una búsqueda sobre una entidad en particular y retorna una única lista de resultados. Si no se especifica ninguna entidad el funcionamiento es similar al endpoint multi-search. Al retornar una única lista de resultados el tamaño de la respuesta es mucho menor. Además, si especificamos que la búsqueda es de tipo “Busca mientras escribes”, la respuesta de Search Service se optimiza aún más, ya que para construir los resultados de la búsqueda rápida se necesita menos información y menos procesamiento de datos.

Este endpoint acepta todos los parámetros definidos en la Tabla 1.

```
/single-search?brand=parkId&entityType=restaurant&q=discount
```

Ejemplo de cómo luce el endpoint single-search.

2.2.3 Respuesta de Search Service

A fines prácticos y para que el lector pueda visualizar cómo funciona Search Service, a continuación se muestra una captura de Pantalla de Postman[12] dónde se realiza una búsqueda utilizando el endpoint *multi-search*.

En el primer recuadro se pueden visualizar los parámetros de búsqueda enviados. Se le pide a Search Service los primeros 20 lugares de comida que contengan el término “restaurant” en el título y que estén dentro del parque solicitado. En el segundo recuadro, se puede ver la respuesta exitosa de Search Service la cual retorna dos listados. El primero con los resultados que cumplen con el criterio de búsqueda y una segunda lista en la cual no se filtra el tipo de entidad, es decir que son resultados que contienen el término de búsqueda “restaurant” pero que no son del tipo especificado. En ambos listados se incluyen el total de resultados disponibles, el tipo de búsqueda, la entidad (si aplica) y un arreglo con los resultados reales provenientes de Elasticsearch. En la captura de pantalla no se ve la estructura que tiene cada uno de los resultados ya que no se considera relevante para explicar el funcionamiento de Search Service. Basta con decir que cada resultado tiene la información que se necesita para crear la vista en la UI: un identificador, un título, una descripción, una imagen, etc.

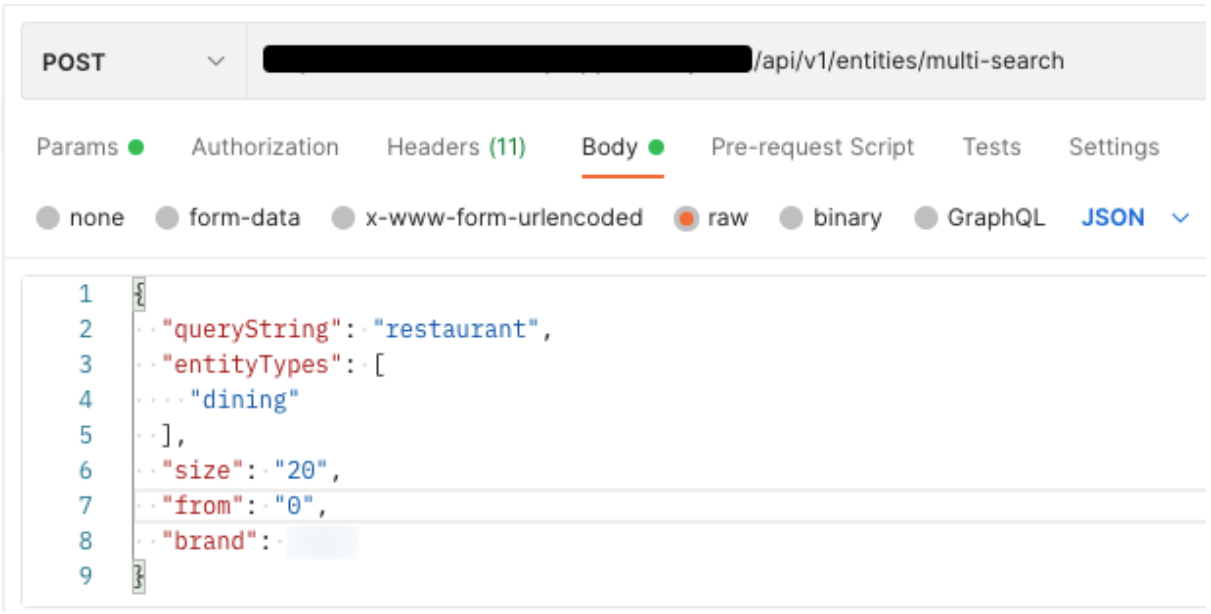

Figura 1: Ejemplo de petición de búsqueda. El campo brand contiene el id del parque solicitado.

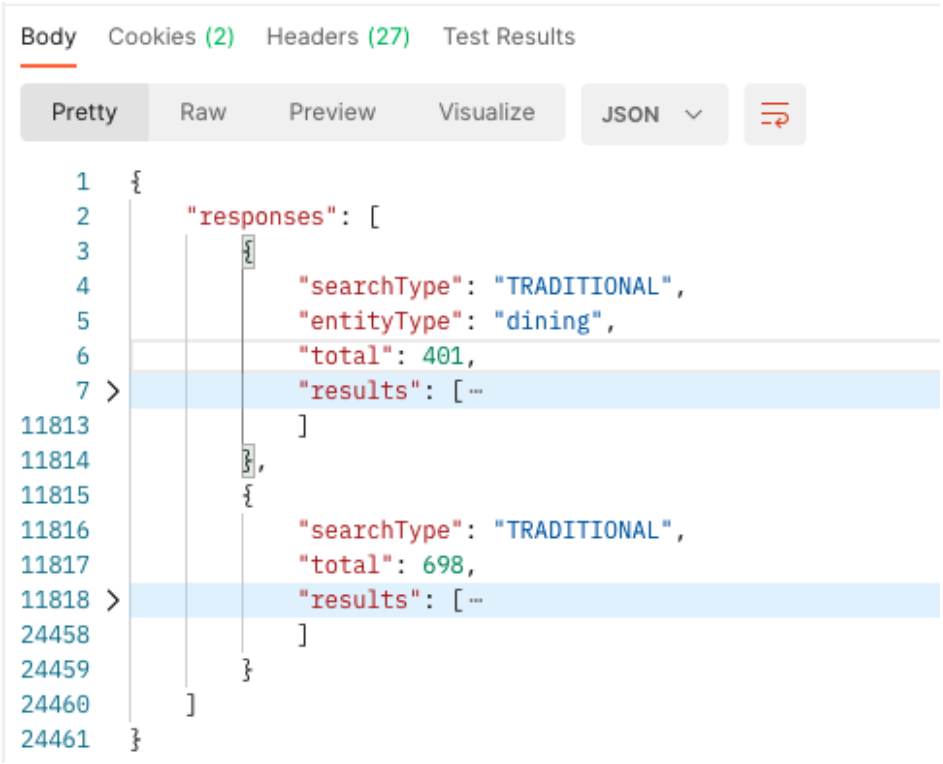

Figura 2: Ejemplo de respuesta exitosa.

Capítulo 3

Arquitectura de la aplicación

Cómo se mencionó en la introducción, la aplicación web sobre la cual se va a desarrollar la abstracción del motor de búsqueda está construida sobre Zend 1.1 siguiendo el patrón de diseño MVC[3] (Modelo - Vista - Controlador). Al ser una aplicación muy grande y de muchos años de desarrollo, la misma cuenta con una gran cantidad de módulos, cada uno con un propósito en particular.

Para que el lector pueda tomar dimensión del tamaño que tiene la aplicación, basta con decir que la misma está dividida en tres repositorios de GIT[15] diferentes. A grandes rasgos se tiene uno para los archivos HTML, CSS y JS, otro para todos los controladores, y por último uno para los distintos modelos de la aplicación. Luego, durante el proceso de ensamblado de la aplicación se toman todos los archivos de dichos repositorios y se crea un único recurso el cual contiene a la aplicación entera y que se utiliza posteriormente al momento de realizar el despliegue.

Esta aplicación como todas las aplicaciones, comenzó siendo pequeña pero con el correr de los años y al recibir tantos requerimientos nuevos por parte del cliente, fue que la aplicación creció hasta el punto en el que se volvió muy difícil de mantener y de escalar. El crecimiento de la aplicación no fue el único problema que se presentó con el pasar del tiempo, también nos encontramos con limitaciones tecnológicas ya que hay ciertas características y funcionalidades que Zend 1.1 no provee ni soporta. Debido a esto, se tuvieron que implementar parches en el framework con funcionalidades que los frameworks de hoy en día ya proveen, más que nada en cuestiones de seguridad algo en lo que el cliente hace constante hincapié. Fue por esto que se decidió comenzar con la migración de la aplicación. Este trabajo es solo una arista de un proyecto aún más grande.

Dejando de lado la historia del proyecto y volviendo a la arquitectura de la aplicación, en la siguiente figura se muestra (de forma muy abstracta) la arquitectura actual y la arquitectura a la cual se quiere llegar con respecto al módulo de búsqueda solamente.

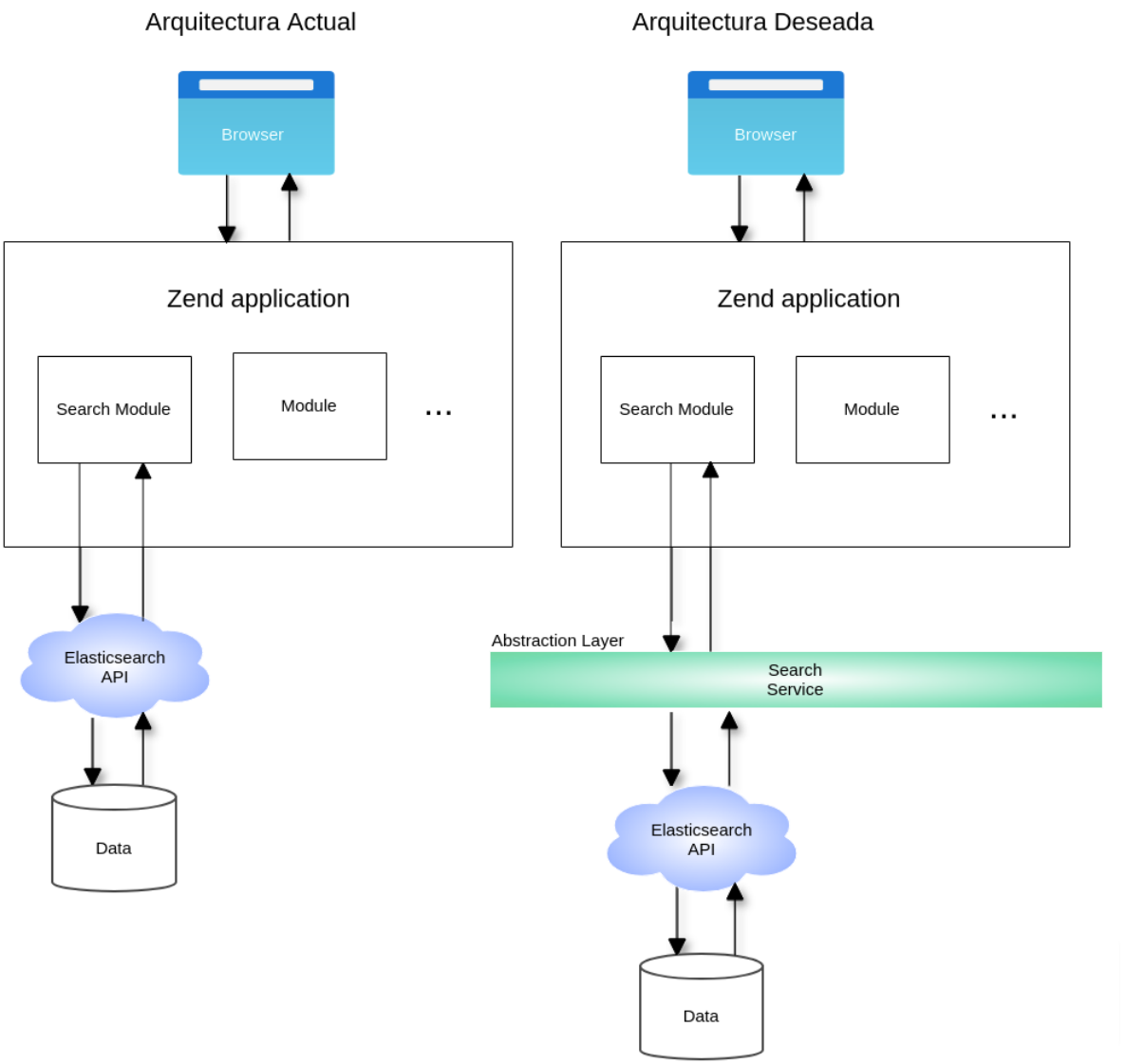

Figura 3: Arquitectura de la aplicación (simplificada)

En la figura de la izquierda podemos ver al Módulo de búsqueda interactuando directamente con la API de Elasticsearch. Esta arquitectura presenta el siguiente problema: ante cualquier cambio o actualización que se tenga que hacer en el Módulo de Búsqueda y que esté relacionado directamente a la API de Elasticsearch va a requerir una nueva salida a producción. En empresas muy grandes, una salida a producción puede llegar a ser realmente costosa. Esta es la principal razón por la cual se optó por abstraer el motor de búsqueda de la aplicación. En la segunda figura vemos al módulo de búsqueda interactuando con Search Service. De esta manera, cualquier cambio que se necesite hacer del lado de Elasticsearch es totalmente transparente para la aplicación web. Search Service es una aplicación totalmente independiente administrada por un equipo reducido de personas por lo que una salida a producción es mucho más rápida, y menos costosa.

El módulo de búsqueda es el lugar donde se va a realizar toda la integración con Search Service. Al estar basado en el patrón MVC dicho módulo va a contar con sus vistas, es decir archivos HTML y CSS, así como también sus archivos JS para construir páginas dinámicas. El módulo también va a contar con su controlador PHP el cual va encargarse de obtener las diferentes configuraciones y de instanciar todas las clases necesarias para consumir la API, para luego poder recibir y atender todas las peticiones de búsqueda que provengan del sitio. El módulo también va a contener todos los modelos de datos los cuales se van a instanciar una vez que se haya obtenido y procesado la respuesta de Search Service.

En el siguiente capítulo se va a detallar cada uno de los archivos que se tuvieron que implementar para poder llevar a cabo la migración así como también el análisis previo a la misma.

Capítulo 4

Comienzo de la migración

Dado que por cuestiones confidenciales de la empresa no se puede mostrar el código entero de la aplicación, se va a describir cómo fue el proceso de desarrollo, las decisiones que se tomaron, y los archivos que se crearon junto con su propósito e importancia de los mismos. Además se van a incluir distintas capturas de pantalla, que ayuden a explicar el funcionamiento de la aplicación.

4.1 Análisis del código legacy

Dado que la funcionalidad de búsqueda ya estaba implementada, se tuvo que realizar un análisis del código legacy para recopilar todas las reglas de negocio y garantizar así, que ninguna funcionalidad se pierda o modifique durante el proceso. La migración tiene que ser totalmente transparente para el usuario final, por lo que este punto es extremadamente importante.

En esta etapa se analizó el controlador principal de búsqueda para determinar los distintos flujos de ejecución, se analizaron también los archivos de configuración existentes los cuales sirvieron entre otras cosas para:

- Saber el número de resultados que se muestran en el listado principal durante la carga inicial.
- El número de resultados que se muestran en el dropdown al realizar una búsqueda del tipo “Busca mientras escribes”.
- Los distintos tipos de resultados que soporta la página: atracciones, eventos, restaurantes, spas, entretenimientos, FAQ, blogs, etc.
- Las reglas específicas que presenta cada tipo de resultado ya que no todos presentan la misma información.
- Y otras reglas de negocio confidenciales, las cuales no se incluyeron en este trabajo.

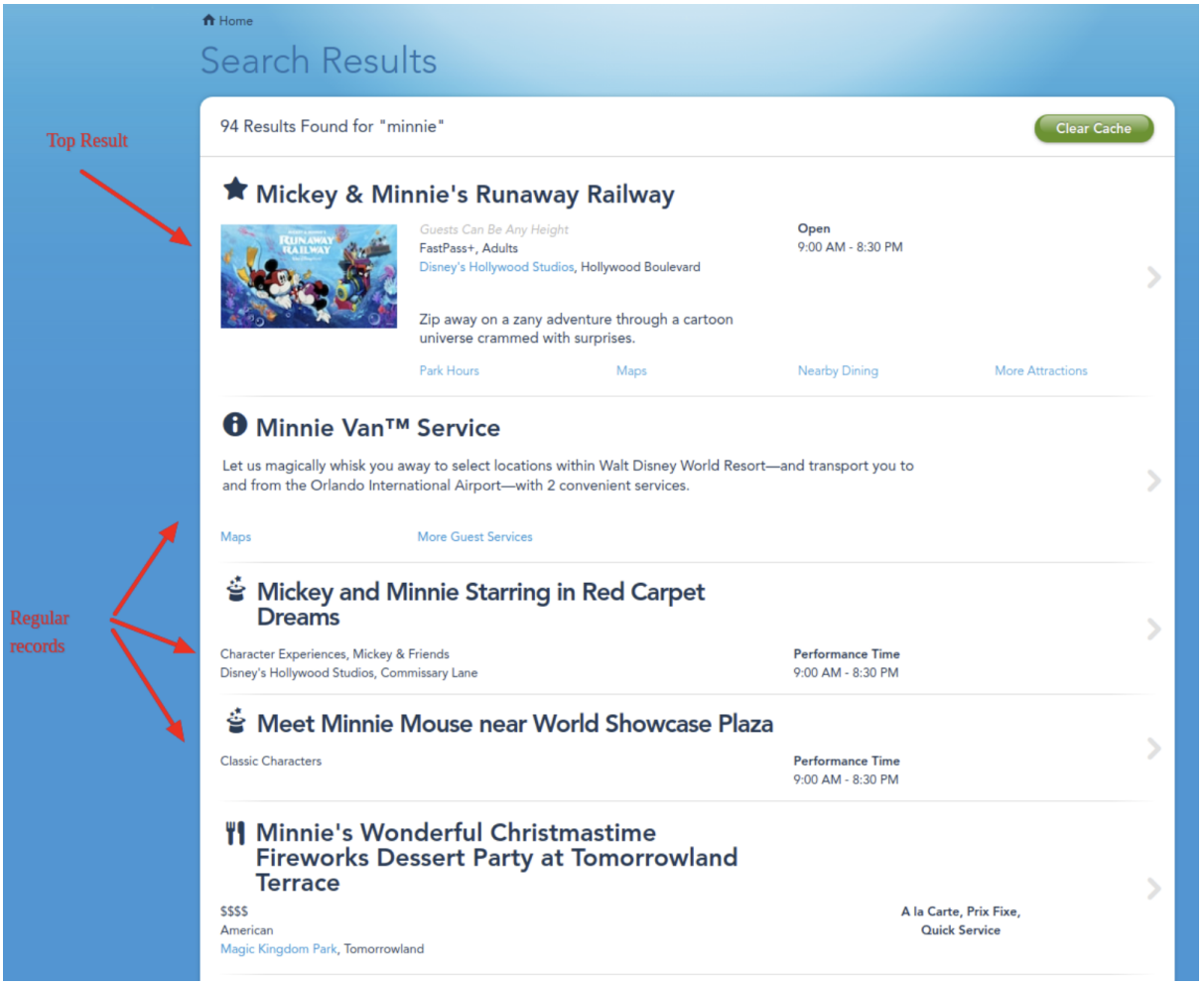

Figura 4: Listado principal de búsqueda.

La captura de pantalla anterior es un claro ejemplo de lo mencionado anteriormente, se pueden ver distintos tipos de resultados (restaurantes, entretenimientos, atracciones, etc) cada uno con características propias. Además se puede ver que el primer resultado de la lista o “top result” como se muestra en la captura contiene información adicional: como por ejemplo una imagen de referencia, una descripción más detallada y varios links de interés.

Dentro de los resultados de búsqueda, también se incluyen las preguntas frecuentes que suelen hacer los usuarios, y los posteos que se hacen en el blog de la empresa.

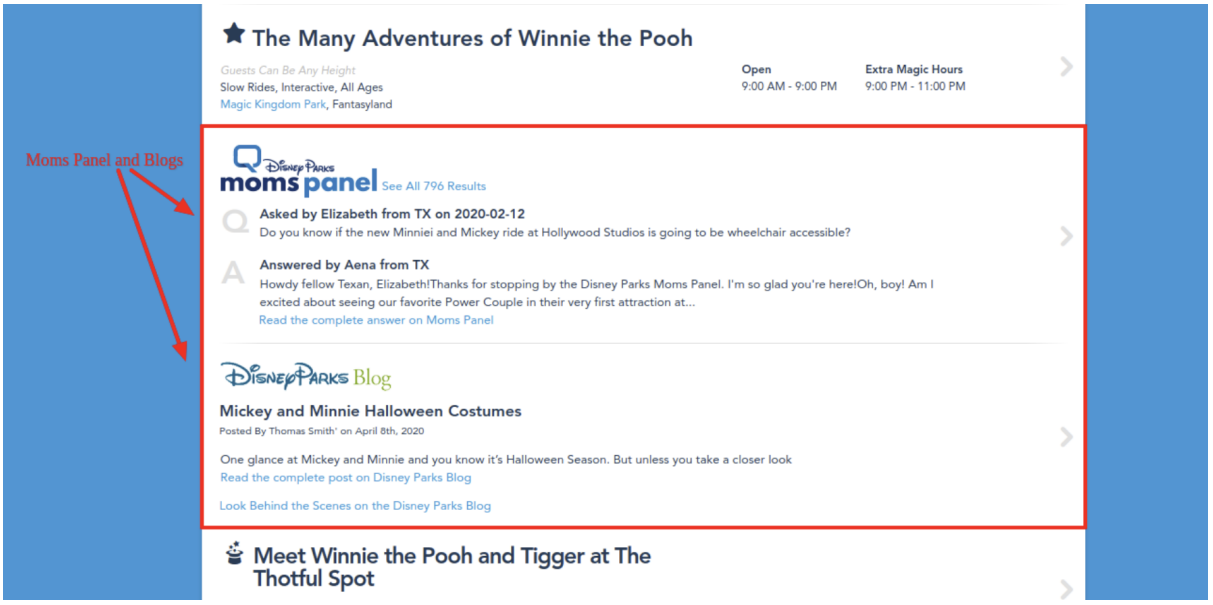

Figura 5: Resultados del tipo: Preguntas frecuentes y Posteos de blog.

El análisis del código sirvió para poder entender de una forma más completa y detallada como es el funcionamiento de la página, desde el momento en el que el usuario realiza una búsqueda hasta cuando se construye el listado de los resultados y se muestra en pantalla. También sirvió para determinar los diferentes tipos escenarios que se pueden presentar como por ejemplo: qué hacer cuando una búsqueda no retorna resultados, o cómo actuar durante una falla del servicio. Para el caso en el que una búsqueda no retorne ningún tipo resultado, se muestra una pantalla especial con algunos consejos para que el usuario pueda volver a buscar. De forma similar se manejan los errores de servicio, donde se redirecciona al usuario a una pantalla de error informando de que hay un problema o informando que el servicio está temporalmente no disponible.

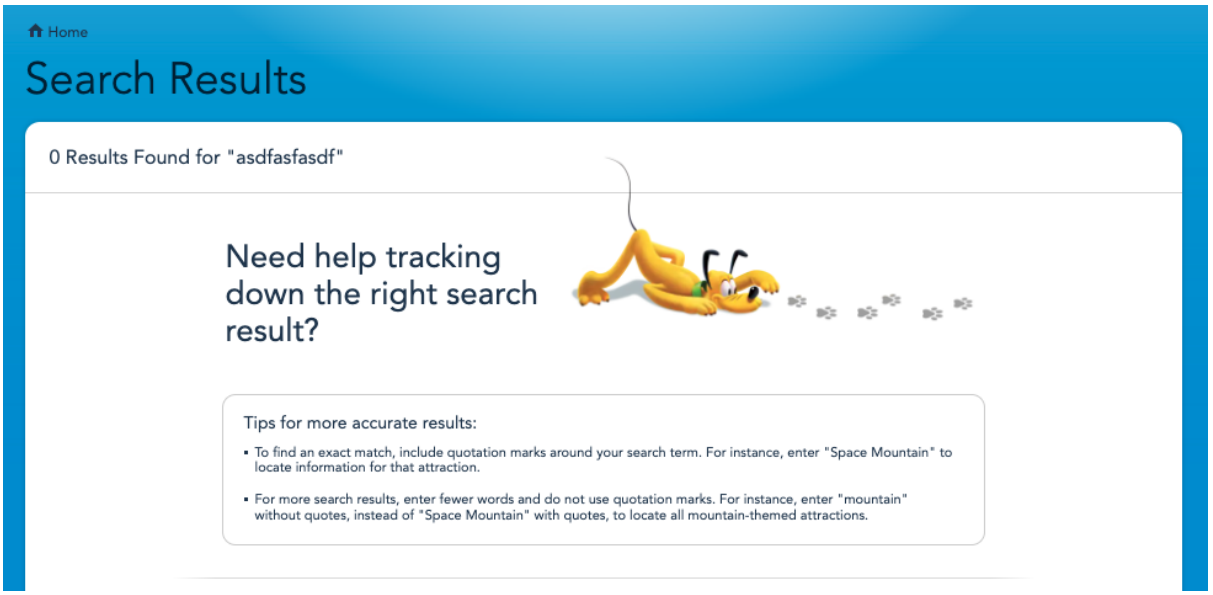

Figura 6: Ejemplo de búsqueda sin resultados.

Cometer fallas en una etapa tan temprana, como lo es la elicitación de requerimientos puede llegar a ser realmente perjudicial. Si no se realiza correctamente se puede perder funcionalidad durante la migración, causando que el resultado final no sea el esperado. De ser así tocaría dar un paso atrás, para poder trabajar sobre los problemas o inconsistencias que se hayan introducido. Por lo que se extendería el proceso de desarrollo, y se incrementarían los costos.

4.2 Search Service Toggle

Al ser una migración y no una implementación que se comienza desde cero, hay archivos que ya están creados y que no necesitamos modificar como lo son los archivos HTML y CSS. Es tan poco el trabajo que se tuvo que realizar para adaptarlos que no los vamos a cubrir en este documento. La intervención realizada comienza a partir del controlador que recibe las peticiones de búsqueda.

Para poder comenzar un trabajo de esta magnitud, el cual se va desarrollando por etapas, primero se tiene que poder garantizar que la funcionalidad de búsqueda actual (Elasticsearch) no se verá afectada durante el proceso. Una vez que se verifica que la implementación nueva funciona correctamente en los ambientes de prueba y posteriormente en producción, recién ahí se da de baja la funcionalidad anterior. En otras palabras, se debe mantener todo el código de Elasticsearch hasta que la integración con Search Service esté finalizada.

Para esto, se creó una configuración en el entorno de desarrollo, la cual se puede habilitar/deshabilitar y que permite cambiar entre los dos flujos de trabajo sin correr ningún tipo de riesgo. Si por alguna razón la nueva implementación falla en producción, lo único que se requiere para restaurar el sitio, es deshabilitar dicha configuración y la aplicación volverá a utilizar Elasticsearch en vez de Search Service como servicio de búsqueda. De esta manera no se requiere hacer lo que se conoce como “rollback”[5] de la salida de producción, ni tampoco se va a necesitar de ningún “hotfix”[6] para solucionar el problema. Ambas situaciones pueden llegar a ser muy estresantes, cuestan dinero, y afectan a la reputación del equipo.

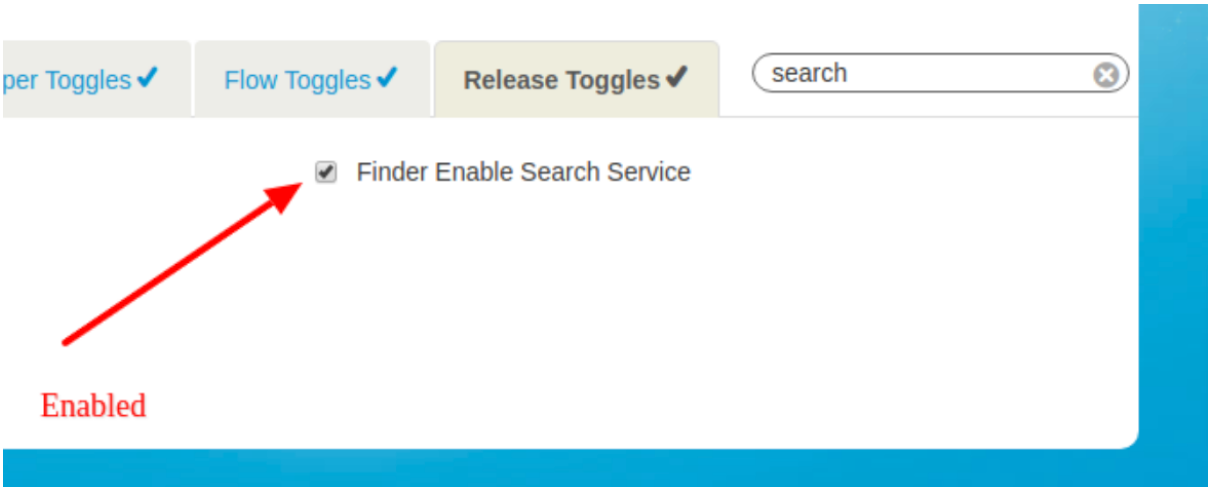

Figura 7: Configuración en el ambiente de desarrollo para habilitar/deshabilitar Search Service.

Desde el punto de vista de la implementación, lo que hace el checkbox que se ve en la captura de pantalla, es setear una cookie en el navegador llamada “Finder Enable Search Service”, la cual se va a mandar en cada petición de búsqueda. Cuando la petición sea interceptada por el controlador, se va a verificar la presencia de la cookie en cuestión y se va a determinar qué flujo de ejecución se va seguir.

En la siguiente figura se muestra de manera abstracta el funcionamiento y la importancia que tiene la cookie en este proceso.

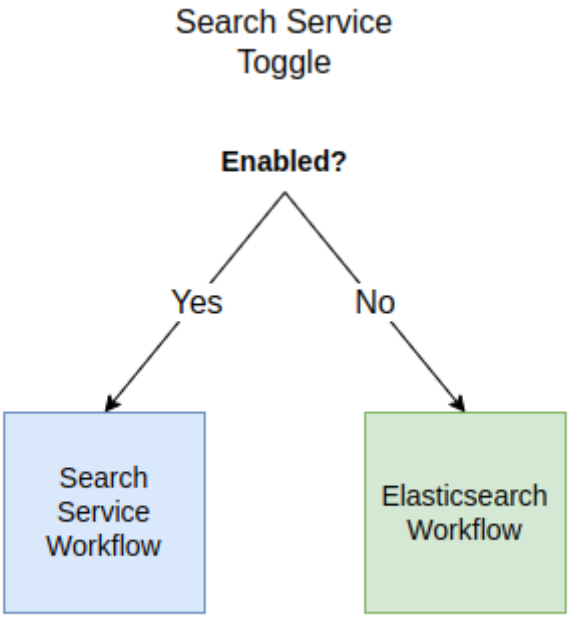

Figura 8: “Search Service Toggle” Configuración que permite determinar el flujo de ejecución.

4.3 Implementación y flujo de ejecución

En este apartado se va explicar cómo es el flujo de ejecución al recibir una petición de búsqueda y parte del trabajo realizado para poder obtener una respuesta exitosa de Search Service. Se explica junto con capturas de pantalla cómo se relacionan cada uno de los archivos involucrados en este proceso. Más adelante se explicará cómo es el procesamiento de la respuesta obtenida y la creación de los modelos de datos.

4.2.1 Search Controller

Este es un controlador ya existente en la aplicación, el cual se encarga de manejar todas las peticiones de búsqueda y de retornar todos los resultados a la UI. Se puede decir que es el punto de partida.

A modo de ejemplo se muestran dos capturas de pantalla de código aislado (el cual no revela ninguna lógica de negocio) en donde se puede ver el lugar exacto en donde se determina qué flujo de ejecución se debe seguir en base al valor que tenga la cookie o toggle mencionado en el apartado anterior (Figura 3). Si la cookie tiene valor verdadero entonces se va a ejecutar la búsqueda utilizando Search Service, en caso de ser falso se va a ejecutar la búsqueda utilizando el método original, que implica llamar a Elasticsearch directamente. Una vez que la migración esté finalizada y se haya probado toda la nueva funcionalidad en producción, se deberá eliminar el flujo de Elasticsearch y se mantendrá sólo el nuevo.

#Llamada al servicio durante la migración.

```
protected function _callService($searchQuery)
{
    $searchResults = null;
    $itemsPerPage = $this->config->search->results->itemsPerPage;

    try {
        if ($this->_searchServiceToggle) {
            // Search Service call
            $searchResults = $this->_searchServiceWorkflow->search($searchQuery);
        } else {
            // Elastic Search Call
            $searchResults = $this->_elasticsearchWorkflow->search($searchQuery, $itemsPerPage);
        }
    } catch (Exception $e) {
        $this->_handleServerErrorResponse();
        return;
    }

    return $searchResults;
}
```

Figura 9: Flujos de ejecución durante la migración. Se puede ver como se agrega Search Service como servicio y cómo se mantiene Elasticsearch al mismo tiempo para evitar cualquier imprevisto.

Llamada al servicio después de la migración.

```
protected function _callService($searchQuery)
{
    $searchResults = [];

    try {
        // Search Service call
        $searchResults = $this->_searchServiceWorkflow->search(
            $searchQuery,
            self::DEFAULT_PAGE_NAME_PARAM,
            self::INDEX_ACTION_PAGE_EVENT
        );
    } catch (Exception $e) {
        $this->_handleServerErrorResponse();
        return;
    }

    return $searchResults;
}
```

Figura 10: Al finalizar la migración queda un único flujo de ejecución, y se hace una limpieza de código en donde se remueve toda la lógica relacionada a Elasticsearch.

4.2.2 Search Service Workflow

Esta es una clase auxiliar, la cual contiene toda la lógica pesada en términos de procesamiento. Es la encargada de obtener todas las configuraciones de búsqueda que aplican a los distintos parques, de instanciar el servicio, y de procesar todos los registros obtenidos de la API de Search Service y dejar solo aquellos que son de interés. Cada registro luego se mapea a un modelo de datos dependiendo su tipo y se retorna una lista de resultados para que sea visualizada en la UI. La estructura de los modelos de datos se explicará en el apartado correspondiente.

Continuando con la Figura 5 y para fines prácticos se muestra una captura de pantalla *parcial*, donde se puede ver cómo se crea el servicio y se realiza una búsqueda.

```php
/**
 * Executes a call to SS
 *
 * @param string $query
 * @param string $pageName
 * @param string $pageEvent
 * @param int $from Offset for the first result.
 * @param boolean $async
 * @return
 */
public function search($query, $pageName, $pageEvent, $from = 0, $async = false)
{
    $size = $this->_config->                    ->size;
    $command = $this->searchServiceCommand->create();

    $search = $command->getSearchCommand(
        $query,
        $pageName,
        $pageEvent,
        $size,
        $from
    );

    $response = $this->getExecutor()->fetch($search);

    if ($response->hasErrors()) {
        $this->_logServiceError($response);
    }
```

Figura 11: Creación del servicio de búsqueda y ejecución del comando multi-search.

En color naranja se muestra cómo se crea/instancia la clase que representa la API de Search Service. En color violeta se muestra cómo se prepara el comando de búsqueda multi-search previo a su ejecución. Es decir, se le envían al comando los parámetros de búsqueda solicitados por el usuario: el término de búsqueda ingresado, el número de registros que se desea obtener y desde qué posición (en caso de que se tenga que aplicar algún tipo de paginación). Finalmente en color violeta se ve la ejecución de dicho comando y la obtención de la respuesta.

En caso de obtener una respuesta exitosa, se procesa la misma, y se crean los diferentes modelos para su posterior uso en la interfaz de usuario. Por cuestiones confidenciales el procesamiento de los registros no fue incluido en la captura de pantalla.

4.2.3 Search Service Command

Como se dijo en el apartado anterior, Search Service Command es una clase PHP, la cual representa al servicio. En este archivo están definidos todos los endpoints de Search Service: multi-search, single-search así cómo también el endpoint que se va a utilizar en la parte de analytics que se verá más adelante.

Esta clase recibe todos los parámetros de búsqueda, setea todos los headers necesarios para realizar la petición (como por ejemplo: Content-Type, Accept-Language, etc) así como también el método HTTP que se va a utilizar (POST). Accept-Language es un header particularmente importante ya que es utilizado por Search Service para retornar resultados de búsqueda en diferentes idiomas. El número de resultados también puede variar por lenguaje ya que no todos los resultados están disponibles para todos los idiomas.

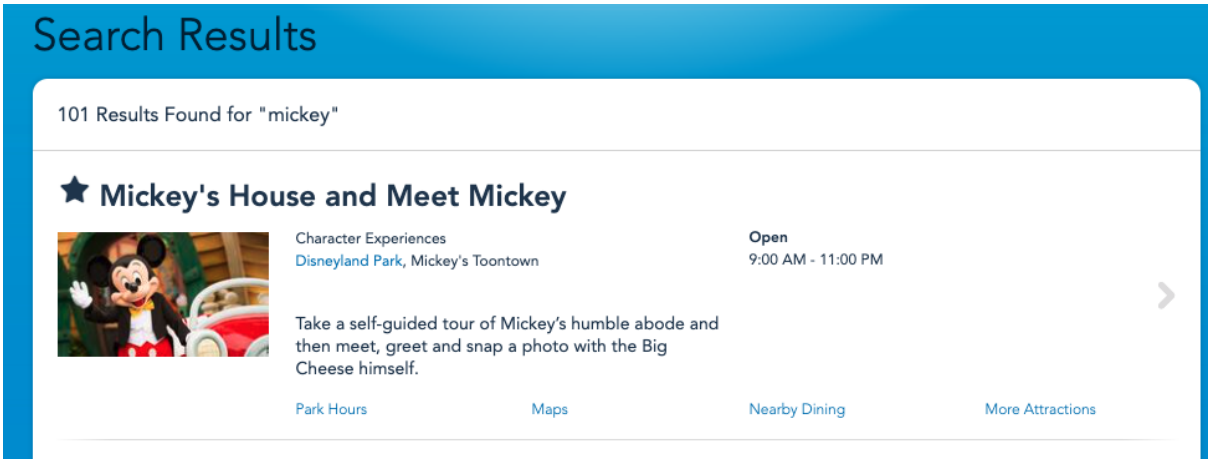

Figura 12: Ejemplo de Búsqueda realizada en inglés

Figura 13: Ejemplo de búsqueda realizada en Español.

El siguiente ejemplo y último de este capítulo, es una continuación de la Figura 6 donde se muestra más en detalle cómo es la preparación del comando multi-search antes de ser ejecutado.

El constructor de esta clase define el método HTTP con el cual se va a realizar la llamada, así como los headers mencionados anteriormente, y toda la información básica que requiere la API de Search Service. Por lo tanto lo que resta es tomar los parámetros recibidos y crear un arreglo que posteriormente se usa para crear el body del request y así completar la petición.

```php
/**
 * Command to search under all entity types.
 * @param string $query is the term to search for
 * @param int $size is the maximum number of records to be retrieved
 * @param int $from is the offset for the first result.
 * @param string $pageName page from where the search was triggered
 * @param string $pageEvent event that triggered the search
 * @return this
 */
public function getSearchCommand($query, $pageName, $pageEvent, $size = 10, $from = 0)
{
    $params = array(
        'queryString' => $query,
        'size' => $size,
        'from' => $from,
        'brand' => $this->_siteId
    );

    $this->setURI($this->_multiSearchUrl);
    $this->_setBodyParams($params);

    return $this;
}
```
Figura 14: Definición del comando de búsqueda multi-search.

De esta forma se da por finalizado el ejemplo. Se pudo observar cómo es el flujo de ejecución al realizar una búsqueda, desde el momento en el que el controlador recibe la petición hasta que se obtiene la respuesta del servicio para luego ser procesada.

Capítulo 5

Modelo de datos

Luego de obtener una respuesta exitosa del servicio de búsqueda, se deben procesar todos los resultados para instanciar los distintos modelos de datos. El listado de búsqueda muestra resultados de todo tipo, atracciones, restaurantes, eventos, preguntas frecuentes, spas, etc. Por lo que para cada tipo de resultado la información que se muestra en la pantalla es diferente. Por ejemplo: para un restaurante se muestra el tipo de servicio brindado, o la especialidad del mismo mientras que para un evento o show se muestran los horarios en los que se puede asistir a la función. Es por eso que se necesita mapear cada resultado a un modelo determinado.

5.1 Mappers

Los mappers son clases las cuales se encargan de recibir un resultado de búsqueda y de extraer todos los atributos que son relevantes para luego instanciar el modelo de datos.

Si bien cada tipo de resultado tiene atributos específicos que no se comparten, también hay otros que son comunes entre todos los resultados sin importar el tipo que sea. Este es un claro ejemplo de lo que se conoce como herencia en la programación orientada a objetos donde se crea una clase padre la cual contiene lógica para extraer todos los atributos y el comportamiento compartido. Y luego cada clase hija es la encargada de definir su comportamiento y extraer los atributos específicos.

Tanto los atributos compartidos, como los atributos específicos para cada modelo de datos están definidos en un archivo de configuración el cual se utiliza para saber qué propiedades deberían extraerse para un resultado dado. Dicho archivo puede actualizarse sin tener que realizar cambios en el código. Se tomó esta decisión para anticipar cambios en la respuesta de Search Service. Por ejemplo: si la propiedad “title” pasara a ser “record-title” bastaría con actualizar dicho archivo para que dicha propiedad se siga extrayendo sin mayores inconvenientes. De esta forma la funcionalidad no se vería afectada y otorgaría cierta flexibilidad ante cualquier tipo de cambio.

En la siguiente figura se muestra la implementación realizada, donde se tiene un mapper genérico el cual se encarga de extraer los atributos comunes a todos los resultados de búsqueda como pueden ser un id, un título, una descripción entre otros y por otro lado están las clases hijas las cuales se encargan de implementar

una función abstracta para setear los atributos específicos al modelo de datos. La función “map” de cada mapper retorna el modelo de datos instanciado.

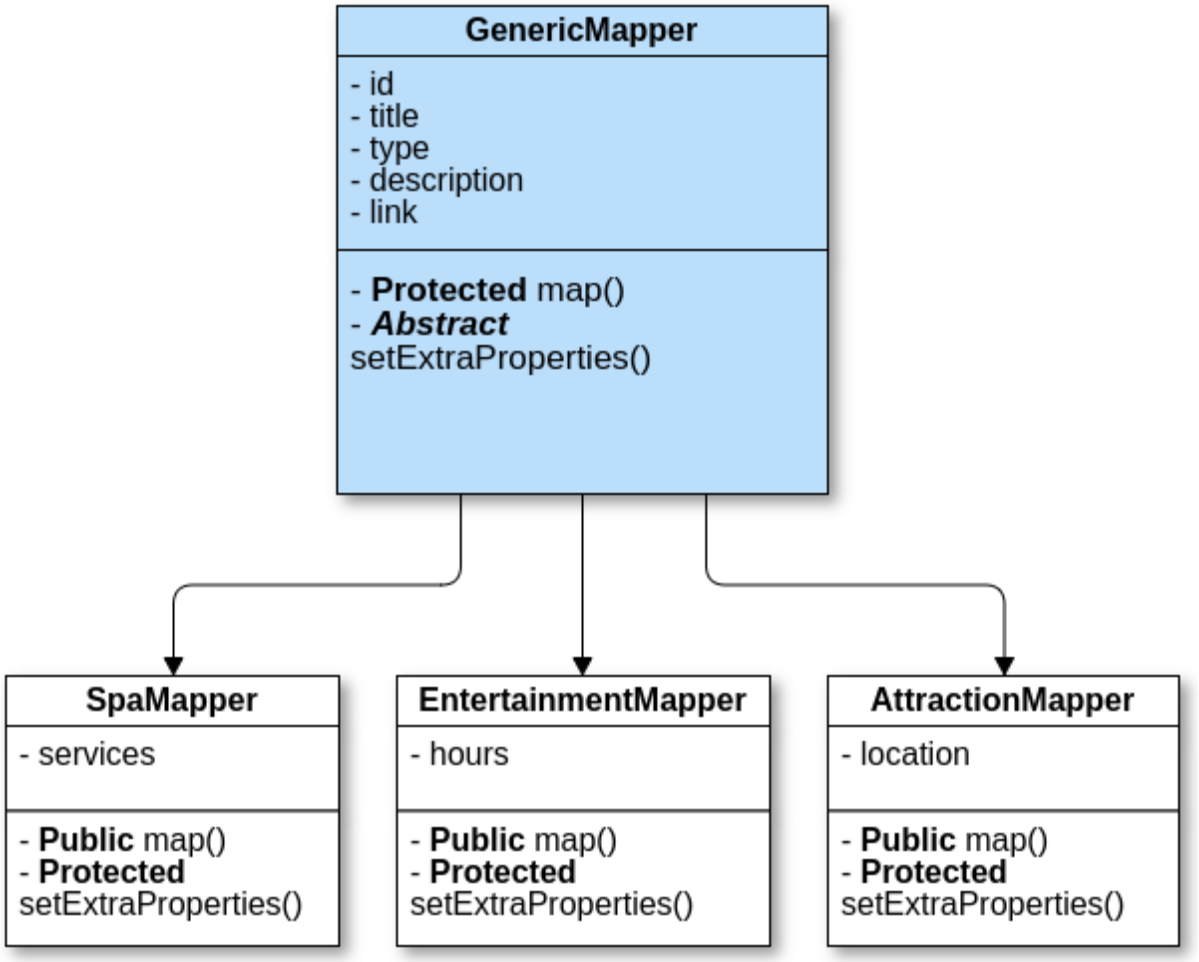

Figura 15: UML que representa los mappers utilizados para la creación de diferentes modelos de datos. Se muestran 3 clases hijas a modo de ejemplo pero hay muchos mappers más dentro de la aplicación.

Nota: en la figura se omitieron ciertas propiedades y métodos que no son relevantes para la explicación del trabajo realizado.

5.2 Procesamiento de los resultados de búsqueda

En el apartado anterior se abordaron los mappers y la responsabilidad que tienen los mismos dentro de la migración. En este apartado y desde un punto de vista más técnico, se mostrará parte de la lógica implementada para el procesamiento de los resultados.

Se implementó una función llamada “mapResults” la cual recibe como parámetro los resultados y se encarga de iterarlos para obtener cada uno de los modelos. Una vez que el modelo de datos fue instanciado es guardado en una lista para ser renderizada en la UI. A continuación se muestra una captura de código para que quede más claro como es el procesamiento.

```php
/**
 * Process the service response and create all the
 * entity models based on each record type.
 * @param stdObject $response Search Service response.
 * @param $offset
 * @return array $results Mapped records.
 */
protected function _mapResults($response, $offset)
{
    $results = [];

    if (sizeof($response->results) > 0) {

        $filteredRecords = $this->_filterRecordsByLocale($response->results);

        // Iterate each record and create each Entity model based on the Record Type.
        foreach ($filteredRecords as $record) {
            $mappedRecord = $this->_getEntityModel($record);
            $results = $this->_saveRecord($mappedRecord, $results, $offset);
        }
    }

    return $results;
}
```

Figura 16: Función encargada de mapear los resultados.

Al finalizar el mapeo de resultados, se retornan los mismos a la UI donde la vista se encarga de recorrerlos y de mostrar el layout correspondiente para cada uno.

Capítulo 6

Listado Principal de Búsqueda

En el Capítulo 4 y 5 se explicó el funcionamiento de cada uno de los archivos que se tuvieron que implementar para llevar a cabo la migración y se incluyeron capturas de pantalla parciales donde se puede ver parte del código implementado. Se describió con palabras el flujo de ejecución, así como también el posterior procesamiento de los resultados y la jerarquía que se utilizó para crear los modelos de datos. A continuación y a modo de cierre se muestra un diagrama de secuencia de alto nivel el cual detalla cómo es el funcionamiento luego de finalizado el trabajo de desarrollo.

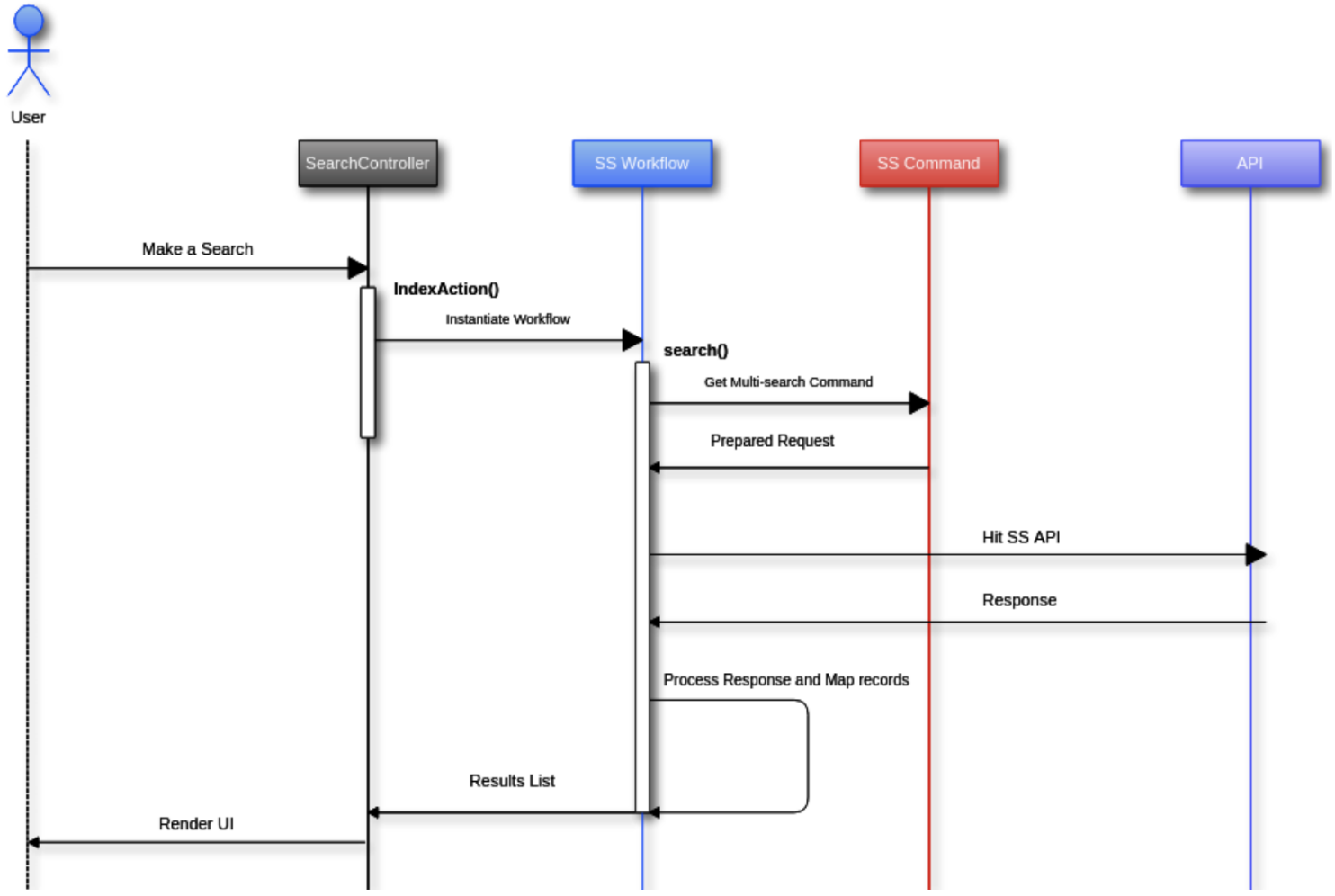

Figura 17: Diagrama de secuencia que muestra el flujo de ejecución durante una búsqueda.

6.1 Verificación de funcionamiento

Todas las páginas de la empresa, tienen un módulo de debug el cual puede ser activado por los desarrolladores para ver información importante sobre sitio como por ejemplo: las cookies que están definidas, el nombre del controlador asociado a la página, así como todos las peticiones que se realizan al servidor tanto POST como GET (cada una con gran detalle).

Este módulo es de gran utilidad a la hora de resolver problemas. Detectar un problema en los servicios del sitio es muy sencillo, solo basta con activar el módulo y ver cuales son las llamadas que están fallando. En este caso en particular se utilizó para verificar que todas las peticiones de búsqueda se realicen sobre Search Service y no sobre Elasticsearch.

En la siguiente captura de pantalla se puede ver como luce una búsqueda exitosa hecha sobre Search Service. Para cada petición se puede ver el código HTTP asociado, el cual indica si el request fue exitoso o no, se puede ver un “trace” es decir un rastro del código que se ejecutó previo a la llamada del servicio, así como los parámetros enviados, la respuesta de Search Service, y otros detalles como por ejemplo los headers que se utilizaron. Y por último vemos el tiempo total que se tardó en obtener los resultados: 312 milisegundos.

Figura 18: Llamado exitoso a Search Service.

Capítulo 7

Barra de navegación y barra secundaria

La barra de navegación es un componente que se encuentra en todas las páginas de la empresa, o por lo menos en su gran mayoría. La misma cuenta con un menú para que el usuario pueda acceder a las funcionalidades más importantes que el sitio tiene para ofrecer, así como también un botón de inicio de sesión y un selector de lenguajes. A su vez, cuenta con un input de tipo texto desde donde el usuario puede realizar búsquedas de todo tipo. El input es creado a través de un plugin de jQuery el cual puede ser inyectado y reutilizado en distintas partes del sitio. En la captura de pantalla se puede ver como el Input se utiliza en dos lugares: la barra de navegación global, y en una barra de búsqueda secundaria.

Así como el listado principal de búsqueda, este plugin de jQuery estaba utilizando Elasticsearch directamente por lo que también tuvimos que agregar migrar dicha funcionalidad.

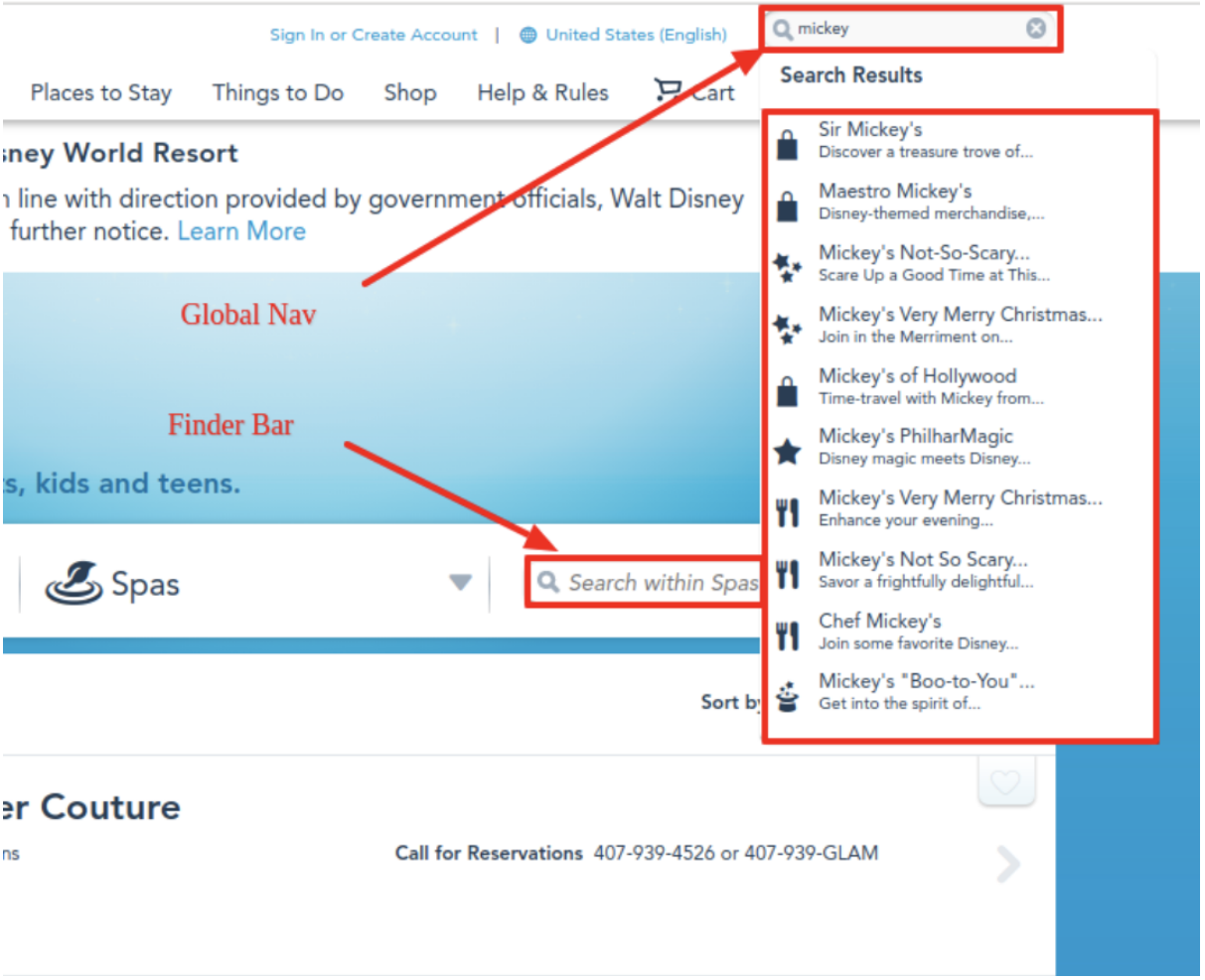

Figura 19: Input de búsqueda.

Como se explicó en el Capítulo 2, para este tipo de búsquedas se utiliza un endpoint diferente al del listado principal el cual retorna una respuesta optimizada ya que para construir el dropdown de resultados se necesita menos información. No se muestran links, imágenes, ni horarios de apertura/cierre de las atracciones y entretenimientos, entre otras cosas.

Además, para evitar llamadas innecesarias a Search Service se agregó un chequeo en el input de tipo texto para controlar que el parámetro ingresado tenga al menos 3 caracteres antes de disparar una búsqueda. Realizar la búsqueda con menos caracteres deriva en que los resultados sean poco certeros. También se limitó el número de resultados mostrados a 10 para mantener un control sobre el tamaño del dropdown. En caso de que el usuario no encuentre lo que está buscando en esos 10 resultados, tiene la posibilidad de apretar la tecla Enter para dirigirse así al listado principal de búsqueda sin tener la necesidad de ingresar la query nuevamente.

7.1 Implementación

El proceso llevado a cabo para migrar esta funcionalidad es similar al del listado, el mismo consta de la creación de una nueva acción dentro del controlador de búsqueda, un nuevo método en Search Service Workflow para el procesamiento de los resultados y un nuevo comando el cual se encarga de recibir todos los parámetros y de preparar la petición a ser ejecutada.

El propósito de cada archivo fue explicado en el Capítulo 4 por lo que en los siguientes apartados sólo se abordarán las incorporaciones que se hicieron en dichos archivos.

7.1.1 Action Controller

En el controlador de búsqueda se agregó un nuevo action para manejar este tipo de búsquedas en particular. Este action controller no renderiza ningún tipo de vista, ya que el mismo se usa para crear el dropdown de resultados de manera dinámica. Es por esto se agregaron validaciones para asegurar que la petición recibida sea por medio del método POST, y se deshabilitaron las funcionalidades nativas del framework para no renderizar absolutamente nada y limitarse a retornar una respuesta en formato JSON.

```php
/**
 * This action handles the call to Search Service. It gets the results to populate
 * the Search Field's dropdown.
 */
public function searchServiceAction()
{
    $request = $this->getRequestWithSanitizedParams();

    if (!$request->isPost()) {
        $this->_handleServerErrorResponse();
    }

    // Disable layout, set no view script rendering
    $this->_helper->layout()->disableLayout();
    $this->_helper->viewRenderer->setNoRender(true);

    $params = $this->_getSearchParamsFromRequest($request);

    $response = $this->_searchServiceWorkflow->getSearchLinks(
        $params->query,
        $params->entityType,
        $params->pageName,
        $params->pageEvent
    );

    return $this->_helper->json($response);
}
```

Figura 20: Acción dentro del controlador de búsqueda, el cual se encarga de recibir las peticiones provenientes del input de búsqueda.

Algo que no se mencionó anteriormente es que al ser un Framework PHP bastante viejo, hay ciertas cuestiones en materia de seguridad que el Framework no provee y que hay que implementar manualmente. Como se puede ver en la captura anterior, a la petición recibida se la sanitiza antes de extraer cada uno de los parámetros de búsqueda. A cada parámetro recibido se le escapan todo tipo de caracteres especiales para evitar inyecciones XSS.

Este tipo de ataques son muy comunes y en general ocurren porque la información ingresada por el usuario no se valida correctamente. El atacante busca inyectar un script o código malicioso a fin de lograr distintos objetivos tales como: robar cookies y sesiones de los usuarios, modificar el sitio web, realizar HTTP requests con la sesión del usuario, redireccionar a los usuarios a sitios dañinos, entre otros. Por eso es tan importante tener estas consideraciones. En Frameworks más nuevos estos aspectos de seguridad ya vienen “out of the box” es decir que son aspectos que ya se tienen en cuenta dentro del Framework por lo que el desarrollador no tiene que hacer nada al respecto.

```php
/**
 * It sanitize the params from the request to avoid XSS/Injection
 * @return Zend_Controller_Request_Http
 */
protected function getRequestWithSanitizedParams()
{
    $request = $this->getRequest();
    $getParams = $request->getParams();
    foreach ($getParams as $param => $value) {
        // We need to check if the value is a string because we also recieve some
        // arrays with information of the device that is making the request.
        if (is_string($value)) {
            $value = $this->view->escape($value);
            $request->setParam($param, $value);
        }
    }
    return $request;
}
```

Figura 21: Función encargada de sanitizar todos los parámetros de búsqueda.

Luego de haber sanitizado todos los parámetros, se extraen y se guardan en un objeto para su posterior uso. La extracción de los parámetros se movió a una función a parte para mantener la simpleza de las funciones y para que la creación de los tests sea relativamente sencilla.

```php
/**
 * Returns an object with all the params needed for making
 * a search request.
 * @param object $request HTTP request
 * @return object $params
 */
protected function _getSearchParamsFromRequest($request)
{
    $params = new stdClass();
    $params->query = $request->getParam(self::SEARCH_QUERY_PARAM);
    $params->entityType = $request->getParam(self::SEARCH_CATEGORY_PARAM, '');
    $params->pageName = $request->getParam(self::PAGE_NAME_PARAM, self::DEFAULT_PAGE_NAME_PARAM);
    $params->pageEvent = $request->getParam(self::PAGE_EVENT_PARAM, '');

    return $params;
}
```

Figura 22: Función encargada de extraer los parámetros de la petición.

7.1.2 Search Service Workflow

“getSearchLinks” es la función que se implementó para obtener los resultados que se utilizarán para crear el dropdown, cada uno de los resultados en el dropdown es un link que apunta a una página de detalle. En la siguiente captura de pantalla se puede observar la creación del comando de búsqueda single-search, el cual se explicó en el Capítulo 2. Al comando se le envía un parámetro extra el cual determina el tipo de búsqueda que se quiere realizar, y que va a permitir obtener resultados optimizados para esta funcionalidad.

```php
/**
 * Executes a call to SS and creates the Finder Links that will be
 * shown in the Finder Bar dropdown and Global nav as well.
 * @param string $query
 * @param string $entityType
 * @param string $pageName
 * @param string $pageEvent
 * @return array $finderLinks
 */
public function getSearchLinks($query, $entityType, $pageName, $pageEvent)
{
    $results = [];
    $size = $this->_config->service->results->dropdown->size;
    $command = $this->searchServiceCommand->create();

    // We set the search-as-you-type flag here to make an optimized search.
    // It has a lighter weight response and a better performance.
    $sayt = true;

    $search = $command->getSingleSearchCommand(
        $query,
        $entityType,
        $pageName,
        $pageEvent,
        $size,
        $sayt
    );

    $response = $this->getExecutor()->fetch($search);
```

Luego de obtenida la respuesta de Search Service, se verifica que la misma no contenga errores, si ocurrió algún error durante la llamada al servicio se registra dicho error para luego tener evidencia de lo ocurrido. El manejo de errores se abordará en el apartado correspondiente.

```
    if ($response->hasErrors()) {
        $this->_logServiceError($response);
    }

    $records = $this->getLinkRecords($response);

    foreach ($records as $record) {
        // Object instantiation moved to a new function for testing purposes.
        $mappedRecord = $this->getSearchServiceLink($record, $query);
        $results[] = $mappedRecord->toArray();
    }

    $finderLinks['links'] = $results;
    $finderLinks['searchId'] = $command->getSearchIdHeader();
    return $finderLinks;
}
```

Figura 23: Función encargada de obtener y de mapear los resultados que se van a mostrar en el dropdown de búsqueda.

Si la llamada al servicio fue exitosa, se procede a extraer los registros. La función “getLinkRecords” está preparada para extraer los resultados sin importar el tipo de búsqueda que se haya realizado ya sea single-search o multi-search. Y siempre retorna el mismo número de resultados, un total de diez registros para la creación del dropdown.

Finalmente por cada registro obtenido se instancia el modelo de datos correspondiente y se crea una lista de resultados la cual es enviada a la UI en formato JSON para su posterior uso.

```php
/**
 * Get the records to build to dropdown links.
 * @param $response
 */
protected function getLinkRecords($response)
{
    $records = [];
    $size = $this->_config->service->results->dropdown->size;

    // Multi Search
    if (isset($response->responses)) {
        $records = $this->getMultiSearchRecords($response);
    }

    // Single Search
    if (isset($response->results) && !empty($response->results)) {
        $records = $response->results;
    }

    $records = $this->_filterRecordsByLocale($records);

    // Make sure only $size records are returned.
    if (sizeof($records) > $size) {
        array_splice($records, $size);
    }

    return $records;
}
```

Figura 24: Función encargada de extraer los resultados de la respuesta de Search Service.

7.1.3 Single Search Command

Por último y para finalizar con la implementación de este tipo de búsquedas dinámicas se muestra como es la creación del comando de búsqueda single-search. Esta función se encarga de recibir todos los parámetros de búsqueda los cuales van a ser parte del body del request a Search Service. Durante la creación del comando de búsqueda también se definen ciertos parámetros los cuales se van a utilizar para el trabajo de analytics. Los mismos se abordarán en el capítulo correspondiente.

```php
/**
 * Command to search under a single entity type.
 * @param string $query is the term to search for
 * @param string $entityType is the entity type used to filter the request
 * @param int $size is the maximum number of records to be retrieved
 * @param string $pageName page from where the search was triggered
 * @param string $pageEvent event that triggered the search
 * @param boolean $sayt Search As You Type flag
 * @return this
 */
public function getSingleSearchCommand($query, $entityType, $pageName, $pageEvent, $size = 10, $sayt = false)
{
    /*
     * Type of search.
     * 'sayt' is used for the search-as-you-type as it is a more performant
     * kind of search and it has a lighter weight response.
     * 'traditional' is the default value for regular searches.
     */
    $type = $sayt ? self::SEARCH_AS_YOU_TYPE : self::TRADITIONAL_SEARCH;

    $params = array(
        'queryString' => $query,
        'size' => $size,
        'brand' => $this->_siteId,
        'type' => $type
    );

    $params = $this->_checkEntityType($entityType, $params);

    $params = $this->_setHistoryParams($params, $pageName, $pageEvent);
    $this->setURI($serviceUri);
    $this->_setBodyParams($params);

    return $this;
}
```

Figura 25: Definición del comando de búsqueda single-search.

7.1.4 Manejo de errores

El manejo de errores en una aplicación es un punto clave, le permite al desarrollador determinar cuál va a ser el comportamiento de la aplicación en caso de que ocurra algún error inesperado. No solo es importante manejar los posibles errores, sino también llevar un registro de ellos para mejorar la aplicación y evitar que vuelvan a pasar.

En todos los flujos de ejecución en los que se llama a Search Service, se controla la respuesta del servicio. En capturas anteriores podemos encontrar ejemplos como el siguiente:

Figura 26: Manejo de errores.

Por lo que en caso de error, se toma la respuesta del servicio, y se extrae el error para luego crear el mensaje que se va a registrar basado en un template que tiene la empresa el cual tiene un formato específico que hace que sea más fácil de buscar entre todos los logs que tiene la aplicación. Por último se arroja una excepción de PHP para que pueda ser capturada y tratada adecuadamente.

```php
/**
 * Logs Search Service errors
 * @param stdObject $response Search Service response
 */
protected function _logServiceError($response)
{
    if (isset($response->getErrorObject()->error)) {

        $error = $response->getErrorObject()->error;
        $message = sprintf(self::ERROR_MSG_TEMPLATE, $error->message);
        $this->_logger->log($message, Zend_Log::ERR);

        throw new PEP_Exception($message);
    }
}
```

Figura 27: Función encargada de registrar los errores provenientes de Search Service.

Por último y a más alto nivel, se envuelve toda la llamada a Search Service dentro de un try/catch para poder capturar cualquier tipo de error que ocurra durante la llamada. En caso de ocurrir, se llama al manejador correspondiente el cual redirecciona al usuario a una página de error informando de lo sucedido. El mismo procedimiento se aplica para el procesamiento de los resultados.

```php
try {
    // Search Service call
    $searchResults = $this->_searchServiceWorkflow->search(
        $searchQuery,
        self::DEFAULT_PAGE_NAME_PARAM,
        self::INDEX_ACTION_PAGE_EVENT
    );
} catch (Exception $e) {
    $this->_handleServerErrorResponse();
    return;
}
```

Manejo de Excepción

Figura 28: Manejo de excepciones.

7.1.5 Verificación de funcionamiento

Para este tipo de llamadas asincrónicas podemos usar las “Devtools” que provee el navegador para verificar que la búsqueda realmente esté siendo servida por Search Service.

Lo primero que se suele verificar es el código HTTP que devolvió la llamada, como por ejemplo un 200 cuando la llamada fue exitosa, o un 500 cuando ocurrió un error en el servidor, entre otros (404, 401, 504, etc). En la siguiente captura se ve el nuevo endpoint que se implementó para obtener los resultados de búsqueda.

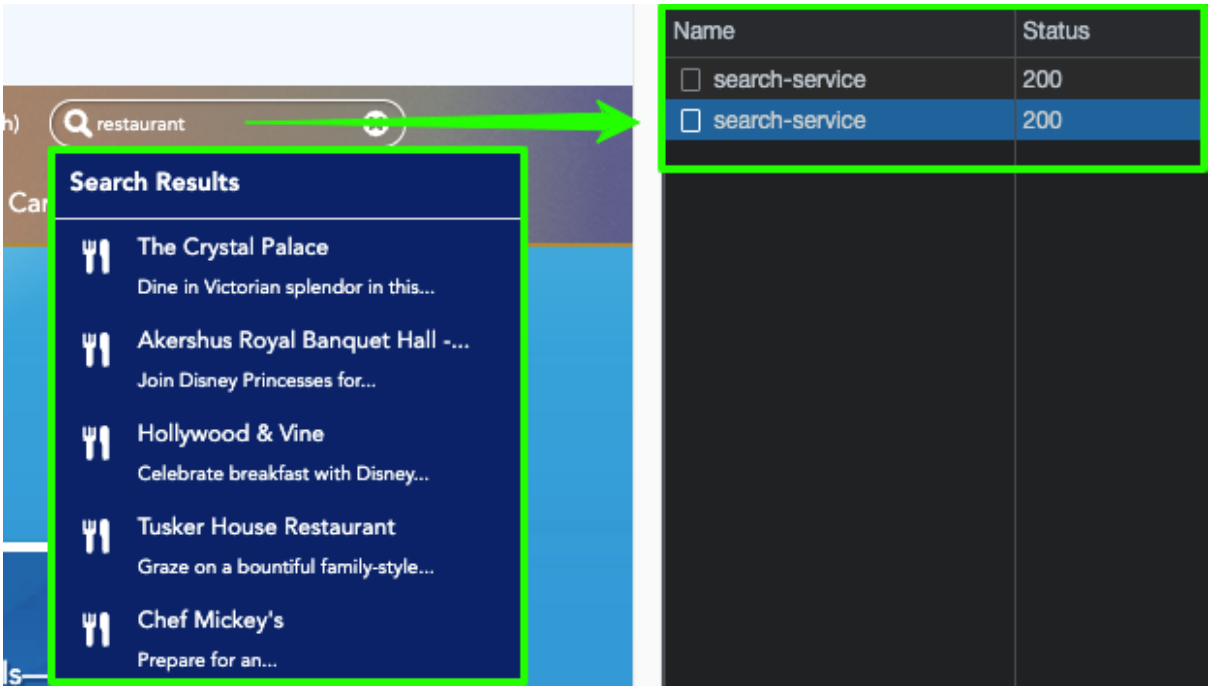

Figura 29: Llamado exitoso a Search Service.

Las Devtools proveen mucha información útil para el desarrollador, como por ejemplo: el tiempo total que tardó la petición de búsqueda en resolverse, los headers que fueron enviados, una vista previa de la respuesta en formato JSON, las cookies que fueron enviadas, entre otras cosas. En la captura de pantalla siguiente se puede observar que el servidor tardó 512 ms en devolver los resultados.

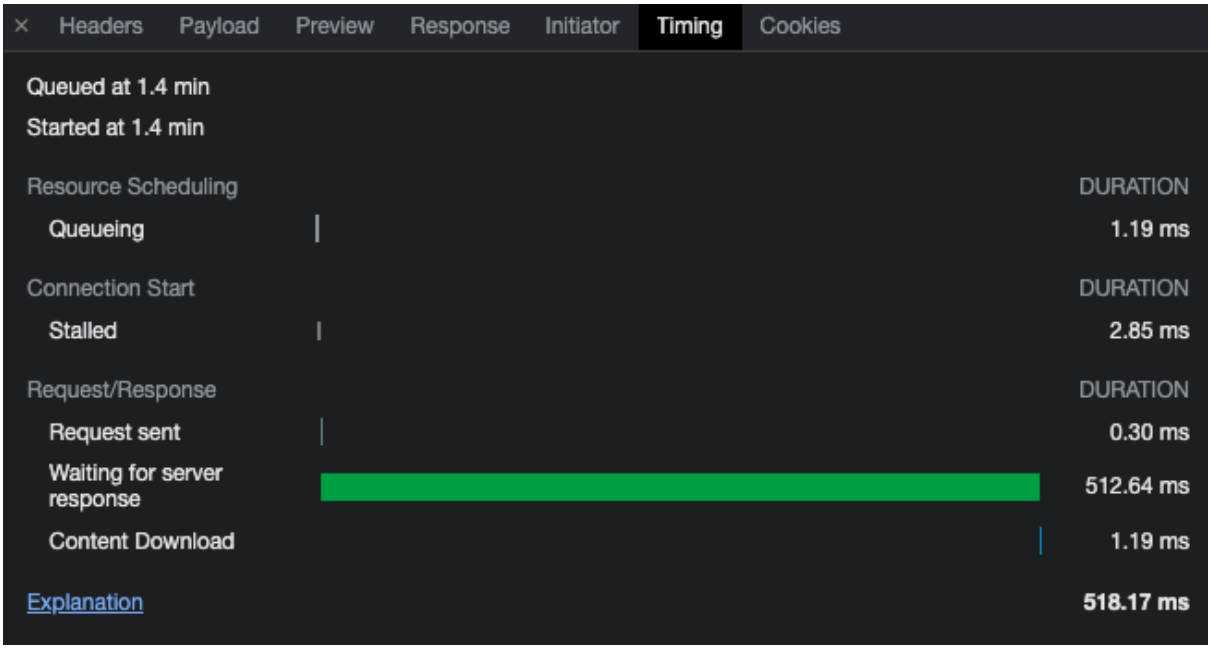

Figura 30: Captura de pantalla la cual muestra el tiempo que tardó Search Service en responder.

Capítulo 8

Analitica de datos

Si bien la abstracción del motor de búsquedas es el tema principal de la Tesina, el trabajo de analítica de datos no es algo menor. Con la incorporación de Search Service se pudo lograr la abstracción deseada, sin modificar la funcionalidad. En cambio la recopilación de toda la información relacionada a las búsquedas realizadas es algo nuevo dentro de la aplicación. Dicha funcionalidad va a brindar información de gran utilidad para que el equipo encargado de mantener a Search Service pueda realizar gráficos en base a todas las búsquedas realizadas y a partir de ahí van a poder tomar decisiones sobre qué cosas se deberían mejorar. El objetivo principal de este trabajo es que sirva para poder mejorar el servicio y que los resultados sean cada vez más certeros.

8.1 Página y lugar desde donde se disparó la búsqueda

Dado que las búsquedas se pueden realizar desde múltiples páginas, dominios y lugares diferentes, lo primero que se debe hacer para comenzar a recolectar información es registrar toda esta información, es decir: desde qué página se disparó la búsqueda, y desde qué lugar ya sea desde la barra de navegación global, la barra de navegación secundaria o si la misma se realizó desde el listado principal de búsqueda. También se tiene que registrar el parque o dominio sobre el cual se realizó la búsqueda, cabe recordar que cada parque dentro de la empresa tiene un dominio web diferente por lo que es muy importante poder diferenciar el tráfico entre dichos dominios.

Implementación

pageName y **pageEvent** son dos parámetros aceptados por los endpoints de Search Service (véase en Tabla 1, Capítulo 2). El primero se va a utilizar para enviar información acerca de la página desde dónde se inició la búsqueda y el segundo parámetro se va a utilizar para enviar información sobre el evento que desencadenó la búsqueda. Ambos parámetros son identificadores que Search Service espera recibir. Dichos identificadores son construidos de la siguiente manera:

A modo de ejemplo, se toma la página en donde los usuarios pueden ver todas las preguntas frecuentes. En la misma se pueden observar dos inputs de búsqueda, es decir que para dicha página hay dos formas distintas de realizar búsquedas hacia Search Service. Utilizando jQuery se obtiene la URL de la página, se limpia el dominio y solo se utiliza el path relativo para construir el pageName. Y luego para cada uno de los inputs se define un pageEvent, es decir un identificador el cual va a

servir para determinar el lugar desde dónde se disparó la búsqueda. Tanto el pageName como el pageEvent son enviados en cada petición a través de AJAX.

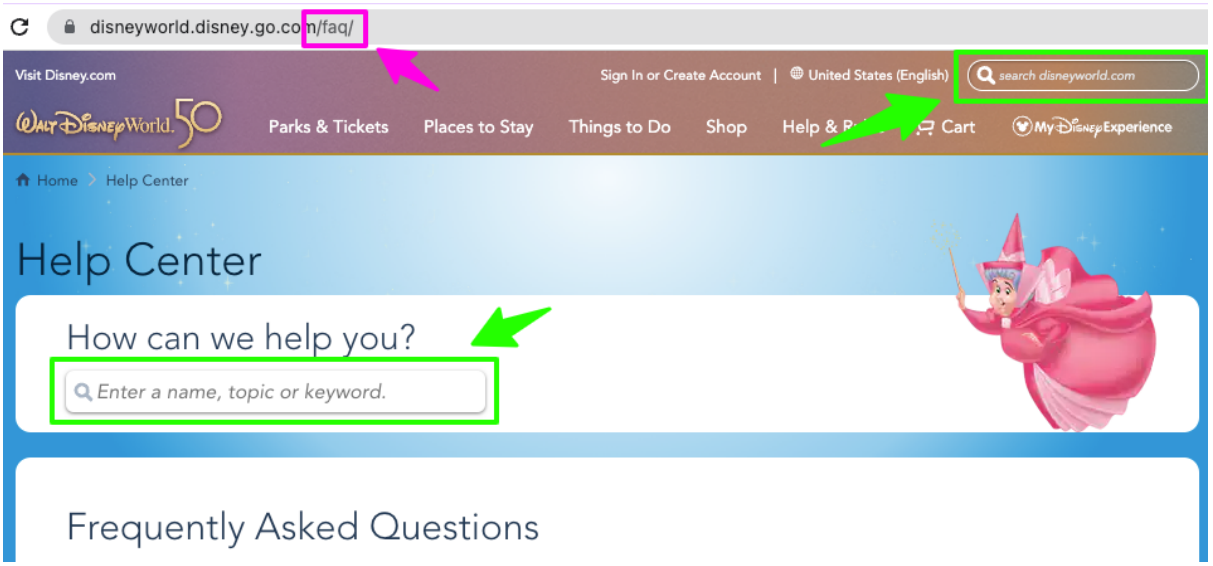

Figura 31: Página de preguntas frecuentes.

Los parámetros son capturados y procesados por el controlador de búsqueda, y luego son enviados al comando de búsqueda independientemente del tipo que sea (single-search/multi-search). Durante la inicialización del comando de búsqueda se utilizan los siguientes templates para crear los identificadores de analytics que Search Service está esperando.

```
; History Data
service.history.pageName = '{SITE_ID}_{PAGE_NAME}'
service.history.pageEvent = '{SITE_ID}_{PAGE_NAME}_{PAGE_EVENT}'
```

Figura 32: Configuración de Search Service la cual contiene los templates utilizados para construir el pageName y el pageEvent requeridos por Search Service.

El {SITE_ID} hace referencia al parque sobre el cual se está realizando la búsqueda. Se utiliza para separar el tráfico entre los distintos dominios web. {PAGE_NAME} y {PAGE_EVENT} fueron explicados al comienzo del apartado.

A continuación se muestra una captura de pantalla la cual muestra cómo se crean los identificadores a partir del template y los parámetros recibidos. Los identificadores son retornados y agregados al body de la petición para ser enviados a Search Service.

```php
/**
 * Set History params
 * @param array $params request params
 * @param string $pageName
 * @param string $pageEvent
 * @return $params
 */
protected function _setHistoryParams($params, $pageName, $pageEvent)
{
    $replacements = [
        '{SITE_ID}',
        '{PAGE_NAME}',
        '{PAGE_EVENT}'
    ];

    $with = [
        strtoupper($this->_siteId),
        ucfirst($pageName),
        $pageEvent
    ];

    $history = $this->_config->service->history;

    $params['pageName'] = str_replace($replacements, $with, $history->pageName);
    $params['pageEvent'] = str_replace($replacements, $with, $history->pageEvent);

    return $params;
}
```

Figura 33: Función encargada de crear los identificadores pageName y pageEvent a partir de las configuraciones vistas en la Figura 31.

8.2 Eficacia del servicio de búsqueda

A partir de este punto Search Service es capaz de identificar cada una de las búsquedas provenientes de la aplicación web. Con esta información se puede determinar con exactitud cuales son las páginas más utilizadas por los usuarios para realizar búsquedas, cuales son los términos más buscados, se pueden crear distintas métricas para saber cual es el sitio con mayor cantidad de tráfico, se puede obtener el número total de búsquedas, el tiempo promedio de respuesta, etc. Al final del capítulo se mostrarán algunos gráficos reales los cuales fueron creados a partir de la información provista por este trabajo.

Como se mencionó anteriormente, Search Service es capaz de identificar cada una de las búsquedas. Ahora bien, como parte de este trabajo también se planteó la posibilidad de medir la eficacia del servicio de búsqueda.

Lo que se consideró como eficacia: En pocas palabras, se puede decir que el servicio de búsqueda es eficaz si lo que el usuario está buscando forma parte de los resultados retornados por Search Service. Por lo que la pregunta qué se debe hacer es: ¿Cómo sabemos si el usuario encontró lo que buscaba? La respuesta es fácil: si seleccionó alguno de los resultados obtenidos.

Desde un punto de vista más técnico, esto significa que hay que establecer una relación entre la búsqueda realizada y los resultados. Para esto Search Service retorna un “Search ID” por cada búsqueda realizada. Dicho ID se asocia a cada uno de los resultados para poder identificar a qué búsqueda corresponde cada uno.

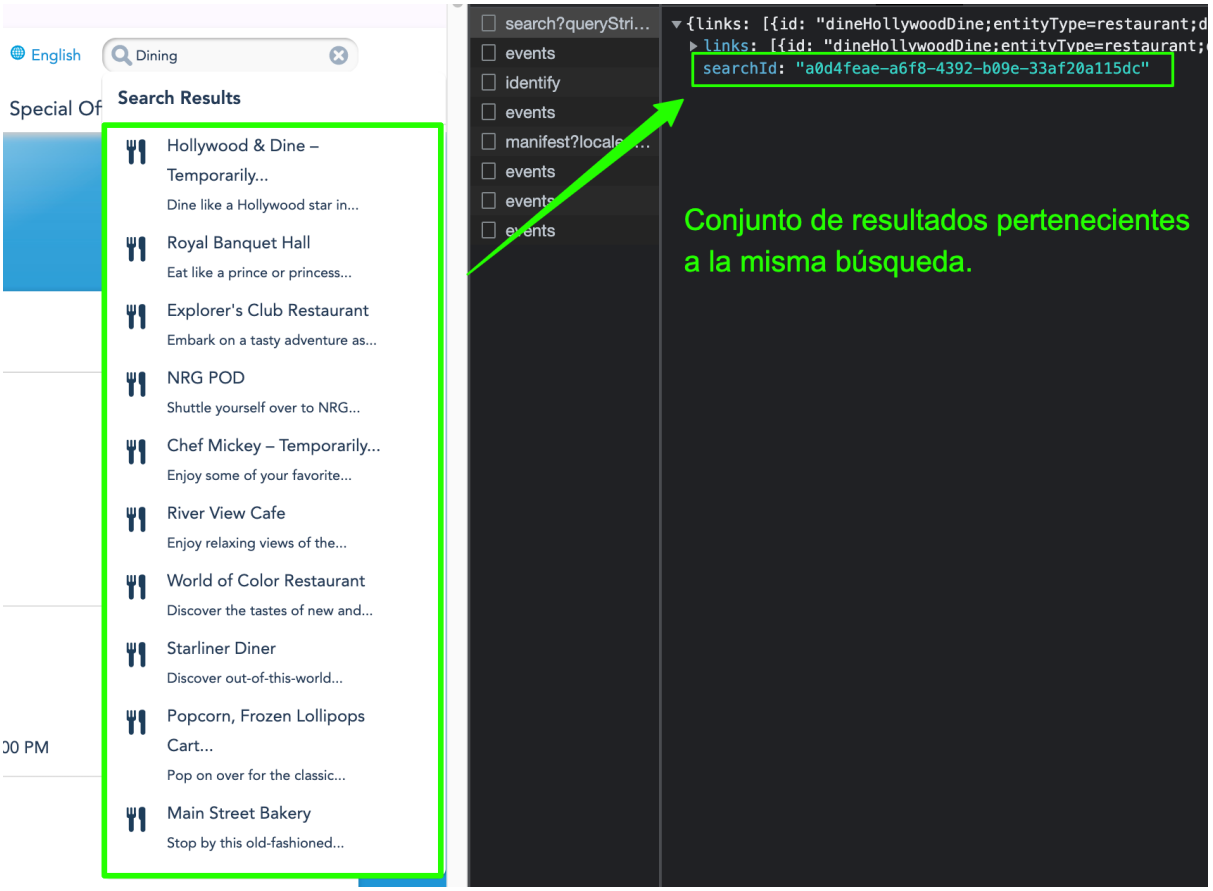

Figura 34: searchId retornado por Search Service.

El mismo concepto aplica al listado principal de búsqueda donde cada uno de los resultados listados tiene asociado un atributo con el Search ID.

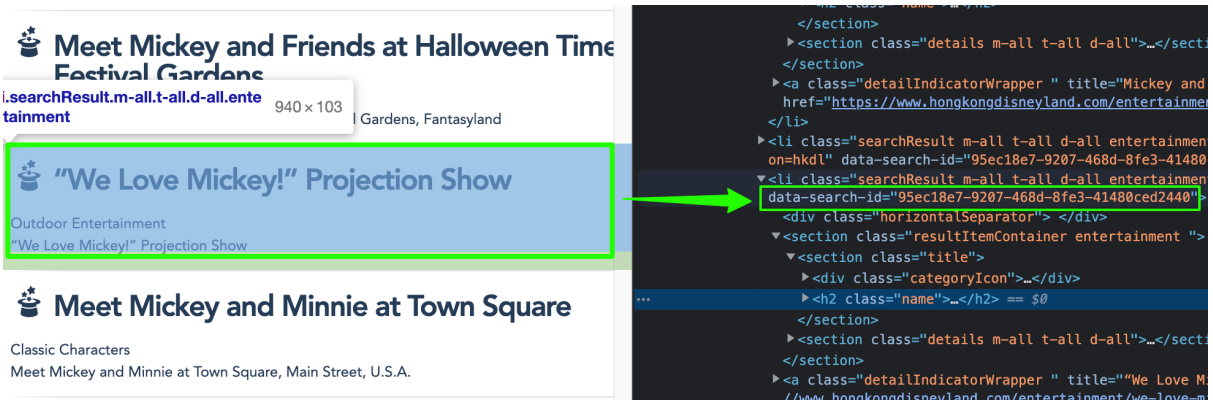

Figura 35: searchId asociado a cada uno de los resultados de búsqueda.

Ahora que existe una relación entre la búsqueda y los resultados que se muestran en la página, lo único que resta es enviar la información de selección hacia Search Service cada vez que el usuario hace click en un resultado.

8.3 Endpoint de Selección

Para registrar las selecciones del usuario, se va a utilizar un endpoint diferente a los vistos anteriormente. Este endpoint se conoce internamente como “Search History”. El mismo espera recibir el Search ID al cual pertenece el resultado seleccionado, el ID del resultado, así como también el título y la posición en la que se encuentra dentro del listado. La posición es muy importante, ya que permite determinar qué tan certera es la búsqueda. No es lo mismo que el usuario seleccione siempre el primer resultado obtenido a que seleccione el último. Si en la gran mayoría de las selecciones se detecta que el usuario selecciona el último o uno de los últimos resultados entonces es claro que la búsqueda no es tan eficiente/certera como se espera.

Nombre	Requerido	Descripción
searchId	si	ID que identifica la búsqueda original que trajo el resultado seleccionado.
documentId	si	ID del resultado seleccionado
position	no	Posición en la que se encontraba el documento seleccionado dentro de la lista de resultados.
title	no	Título del documento seleccionado

Tabla 2: Lista de algunos parámetros admitidos por el endpoint de selección.

Nota: este endpoint también recibe información acerca de la sesión del usuario. Esto permite detectar todas las búsquedas que se realizaron durante una misma sesión. Puede servir como punto de comparación en caso de que el usuario tenga que realizar múltiples búsquedas para encontrar lo que estaba buscando. Los parámetros relacionados a la sesión del usuarios se omitieron en la tabla ya que es información sensible y no hace al trabajo.

8.4 Implementación

A todos los resultados de búsqueda ya sea del listado principal, o del drop down se les asocia un manejador de jQuery para el evento “click”. El mismo se encarga de extraer toda la información del ítem/resultado seleccionado y de realizar una petición AJAX al endpoint de selección.

A continuación y a modo de ejemplo se muestra una captura de pantalla con el código del manejador implementado para el dropdown de resultados. De manera similar se implementó el manejador para el listado principal.

```
/**
 * Send Selection information to Search Service.
 * @param {object} item Dropdown search result
 */
handleSearchSelection: function (item) {
    $.ajax({
        url: linkGenerator.build('/search/search-history'),
        async: true,
        type: 'POST',
        dataType: 'json',
        data: {
            title: item.fullTitle,
            position: item.position,
            documentId: item.id,
            searchId: item.searchId
        }
    });
},
```

Figura 36: Manejador de jQuery encargado de enviar la información de selección a Search Service.

Luego, en el controlador de búsqueda (Search Controller) se agregó una nueva acción, la cual se encarga de sanitizar todos los parámetros recibidos para luego enviarlos al Workflow de Search Service de manera segura.

```php
/**
 * This action sends History data to Search Service.
 */
public function searchHistoryAction()
{
    $request = $this->getRequestWithSanitizedParams();

    if (!$request->isPost()) {
        $this->_handleServerErrorResponse();
    }

    // Disable layout, set no view script rendering
    $this->_helper->layout()->disableLayout();
    $this->_helper->viewRenderer->setNoRender(true);

    // Get Request params
    $id = $request->getParam(self::SEARCH_RESULT_DOCUMENT_ID);
    $title = $request->getParam(self::SEARCH_RESULT_TITLE);
    $position = $request->getParam(self::SEARCH_RESULT_POSITION);
    $searchId = $request->getParam(self::SEARCH_RESULT_SEARCHID);
    $pageName = $request->getParam(self::PAGE_NAME_PARAM);

    $this->_searchServiceWorkflow->sendHistoryData($id, $title, $position, $searchId, $pageName);
}
```

Figura 37: Acción dentro del controlador de búsqueda encargado de recibir la información de selección.

En la clase encargada de manejar los diferentes flujos de ejecución (Search Service Workflow) se agregó un nuevo método llamado “sendHistoryData”, el cual se encarga de validar los parámetros recibidos. Si el Search ID recibido es válido, y si el resultado seleccionado pertenece a dicha búsqueda entonces se inicializa y se ejecuta un nuevo comando llamado “getSearchHistoryCommand” el cual se implementó para enviar toda la información de selección a Search Service.

Para dar por finalizado este apartado, se muestran dos capturas de pantalla con la lógica mencionada anteriormente: en la primera captura se ven las validaciones previas a la ejecución del comando, y en la segunda captura se ve la definición del comando en sí.

En este punto el equipo encargado de gestionar el servicio de búsqueda cuenta con todo lo necesario para crear los distintos gráficos. En el apartado siguiente y el último del capítulo se muestran algunos gráficos creados a partir de la información proporcionada.

```php
/**
 * Sends History Data to Search Service.
 * @param $documentId Selected result ID
 * @param $title Selected result Title
 * @param $position Zero-based index of the selected search result.
 * @param $searchId searchId header value.
 * @param $pageName Page Name
 */
public function sendHistoryData($documentId, $title, $position, $searchId, $pageName = '')
{
    $command = $this->searchServiceCommand->create();
    $session = $this->_session;
    $validPageName = isset($session->pageName) && $session->pageName === $pageName;

    // For search-service-listing we don't receive the $searchId, instead we receive the pageName
    // and then we get the searchId from session and validate the given $documentId.
    if ($validPageName && $session->validSearchId && in_array($documentId, $session->searchRecords)) {
        $searchId = $this->getSearchId();
        $session->validSearchId = false;
    }

    // Just a safe in case the searchId is missing.
    // We don't want to send selection data without a search associated.
    if (empty($searchId)) {
        return;
    }

    $history = $command->getSearchHistoryCommand(
        $title,
        $position,
        $documentId,
        $searchId
    );

    $response = $this->getExecutor()->fetch($history);

    if ($response->hasErrors()) {
        $this->_logServiceError($response);
    }
}
```

Figura 38: Función encargada de validar la información de selección y de ejecutar el llamado a Search Service.

```php
/**
 * Command to send History Data to Search Service
 * @param string $title Document Title
 * @param integer $position Zero-based index of the selected search result
 * @param string $documentId Document ID
 * @param string $searchId Last search-id header value
 * @return this
 */
public function getSearchHistoryCommand($title, $position, $documentId, $searchId)
{
    // Clean up the title and decode HTML entities.
    // Input:  Disney Floral &amp;amp; Gifts
    // Output: Disney Floral & Gifts
    $title = html_entity_decode(strip_tags($title));
    $params = array(
        'title' => $title,
        'searchId' => $searchId,
        'position' => $position,
        'documentId' => $documentId
    );

    $this->setURI($this->_historyUrl);
    $this->_setBodyParams($params);

    return $this;
}
```

Figura 39: Definición del comando Search History.

8.5 Gráficos y reportes

Gracias a toda la información proporcionada por este trabajo, el equipo de Search Service pudo construir un gran número de gráficos para obtener estadísticas sobre el funcionamiento del servicio y de las búsquedas del sitio en general. Podemos dividir estos gráficos en dos grupos: aquellos que están relacionados al funcionamiento del servicio y aquellos que están relacionados o que proveen información sobre cada uno de los parques temáticos de la empresa.

Cada uno de los gráficos creados son configurables en varios aspectos de los cuales vamos a mencionar sólo dos: por dominio web, es decir que podemos ver los gráficos para cada uno de los parques que tiene la empresa y realizar comparaciones entre los mismos. Y la segunda, y quizás una de las más utilizadas, es la posibilidad de ver la información para un rango de fechas dado. Si la fecha corresponde al día de hoy hay un subconjunto de configuraciones que se pueden utilizar como por ejemplo: últimos 15 minutos, última hora, últimas 4 horas, etc. Esto sirve por ejemplo para analizar el número de búsquedas durante una fecha especial o durante algún evento importante.

De todos los gráficos y estadísticas que se tienen sobre el servicio, sólo se abordarán aquellas que tengan una relación más directa con el trabajo realizado.

Nota: Todas las capturas de pantalla mostradas a continuación son estadísticas registradas durante un lapso de **60** minutos.

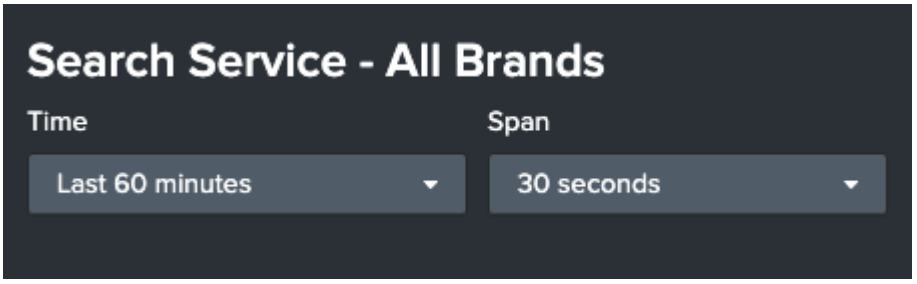

Figura 40: Configuración de tiempo utilizada para la creación de los gráficos.

8.5.1 Gráficos relacionados al servicio en general

En este grupo se encuentran algunos de los gráficos que se crearon y que están directamente relacionados al servicio como por ejemplo: el número de búsquedas realizadas, el tiempo de respuesta promedio el cual es muy útil, por ejemplo, cuando se reciben reportes diciendo que la página de búsqueda tarda mucho en cargar. Con solo ver el indicador se puede descartar si el problema está relacionado a Search Service o si está siendo causado por otra cosa. También hay gráficos que muestran la tasa de errores del servicio, y todos los códigos de respuesta retornados por el mismo, entre otros.

- Número de búsquedas:

Muestra todas las peticiones de búsquedas recibidas sin importar el tipo que sea (Búsquedas hechas desde la barra de búsqueda global, la barra secundaria, el listado principal, etc).

Figura 41: Número total de búsquedas registradas en un lapso de 60 minutos.

- Tiempo de respuesta promedio:

Es un indicador del tiempo promedio que tarda Search Service en responder una petición de búsqueda. El mismo está expresado en milisegundos.

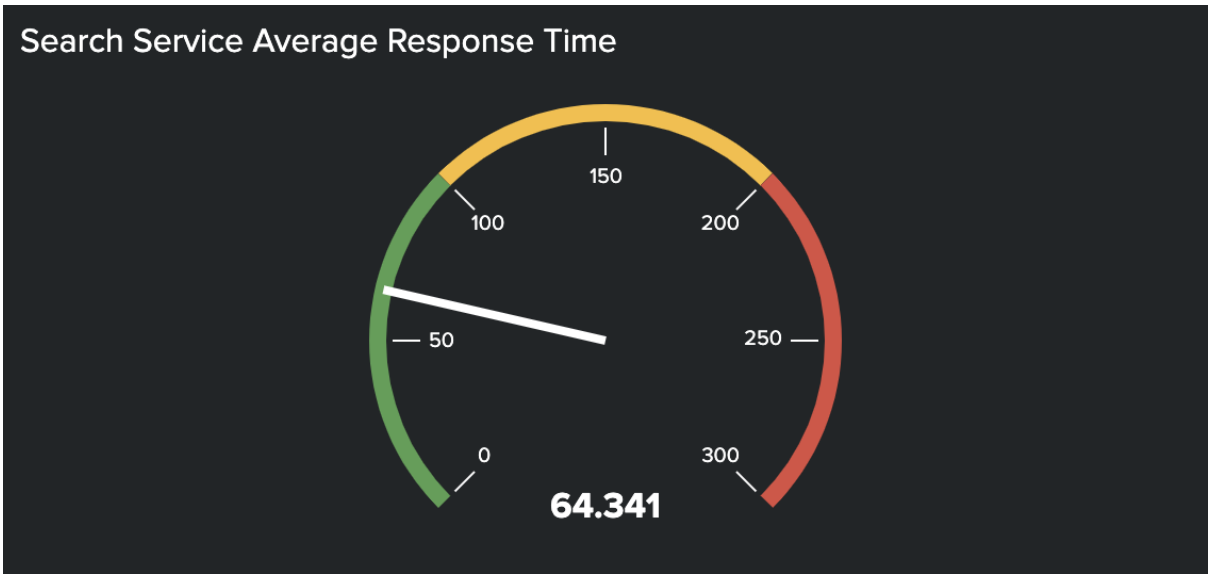

Figura 42: Tiempo de respuesta promedio en un lapso de 60 minutos.

- Número de peticiones de búsquedas recibidas por plataforma:

El siguiente gráfico en forma de torta representa el porcentaje de peticiones provenientes de cada una de las siguientes plataformas: iOS, Android y Web. El 71% de las peticiones fueron realizadas desde dispositivos con iOS. Mientras que en menor medida le siguen los dispositivos Android y por último la web. 17% y 12% respectivamente.

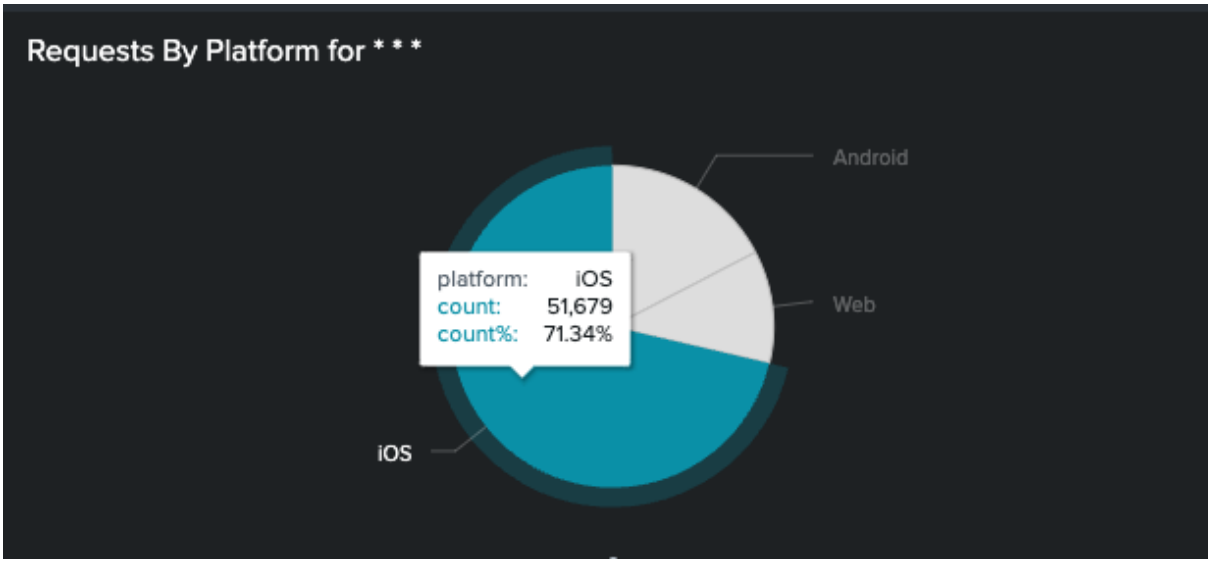

Figura 43: Número de peticiones recibidas por plataforma en un lapso de 60 minutos.

- Códigos de respuesta y tasa de errores:

Estos gráficos llevan un registro de la tasa de errores, y de los códigos de respuesta retornados. Al momento que se obtuvieron las capturas de pantalla se puede ver que el servicio estaba funcionando con total normalidad, solo se ven códigos de respuesta 200 OK, y ningún error 500 o 400. Estos gráficos son útiles ante algún fallo en el servicio.

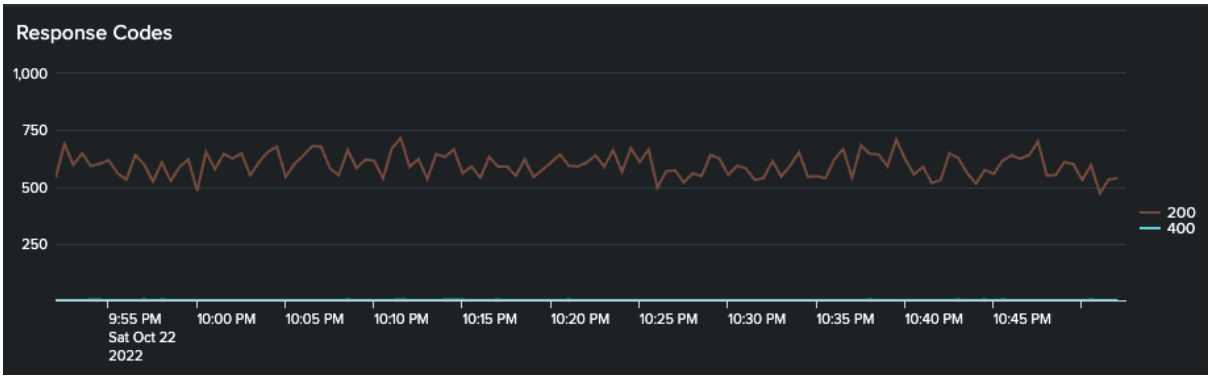

Figura 44: Codigos de respuesta retornados en un lapso de 60 minutos.

Figura 45: Tasa de errores registrada en un lapso de 60 minutos.

8.5.2 Gráficos relacionados a cada uno de los parques temáticos

En este segundo grupo se encuentran los gráficos que están directamente relacionados al trabajo de analytics y que no se hubieran podido construir de no haberse llevado a cabo.

A partir de estos gráficos se pudo obtener información de gran valor para la compañía, teniendo en cuenta que la aplicación tiene más de 10 años de vida y que por primera vez se está analizando la forma en que los usuarios realizan las búsqueda, qué buscan, y desde dónde lo buscan. Todo esto permitió que se pueda mejorar el servicio de manera significativa. A continuación se detallan algunos ejemplos de los beneficios conseguidos por este trabajo:

- Términos más buscados por los usuarios:

Muestra de una forma muy intuitiva cuáles son los términos más buscados por los usuarios. Este gráfico es de gran utilidad, ya que gracias a él se pudo detectar que muchas veces los usuarios ingresan los términos mal escritos como por ejemplo “themepark” en vez de “theme park”. Ante la recurrencia de este tipo de errores se tomó la decisión de crear una mejora dentro del servicio de búsqueda. La misma consiste en que para un resultado en particular existan distintos “sinónimos” los cuales se utilizan para retornar el mismo resultado. De esta manera, el usuario es capaz de encontrar exactamente lo que está buscando más allá de haber realizado la búsqueda con errores de ortografía, sin espacios, etc.

Figura 46: Términos más buscados por los usuarios en un lapso de 60 minutos.

- Gráfico de selección:

Como se vio en el apartado 8.3, cada vez que se selecciona un resultado de búsqueda, se envía información sobre la posición en la que se encuentra el mismo dentro de la lista de resultados obtenidos.

El siguiente gráfico fue construido en base a esa información, el mismo muestra que en la gran mayoría de las veces el usuario selecciona el primer resultado o el segundo, lo cual quiere decir que en general el usuario encuentra justo lo que estaba buscando dentro de los primeros resultados. Si por el contrario, el gráfico de torta revelara que los resultados más seleccionados son los últimos de la lista, entonces tocaría revisar el funcionamiento del servicio para determinar la causa de dicho comportamiento y en base a eso tomar ciertas decisiones que permitan mejorar el servicio.

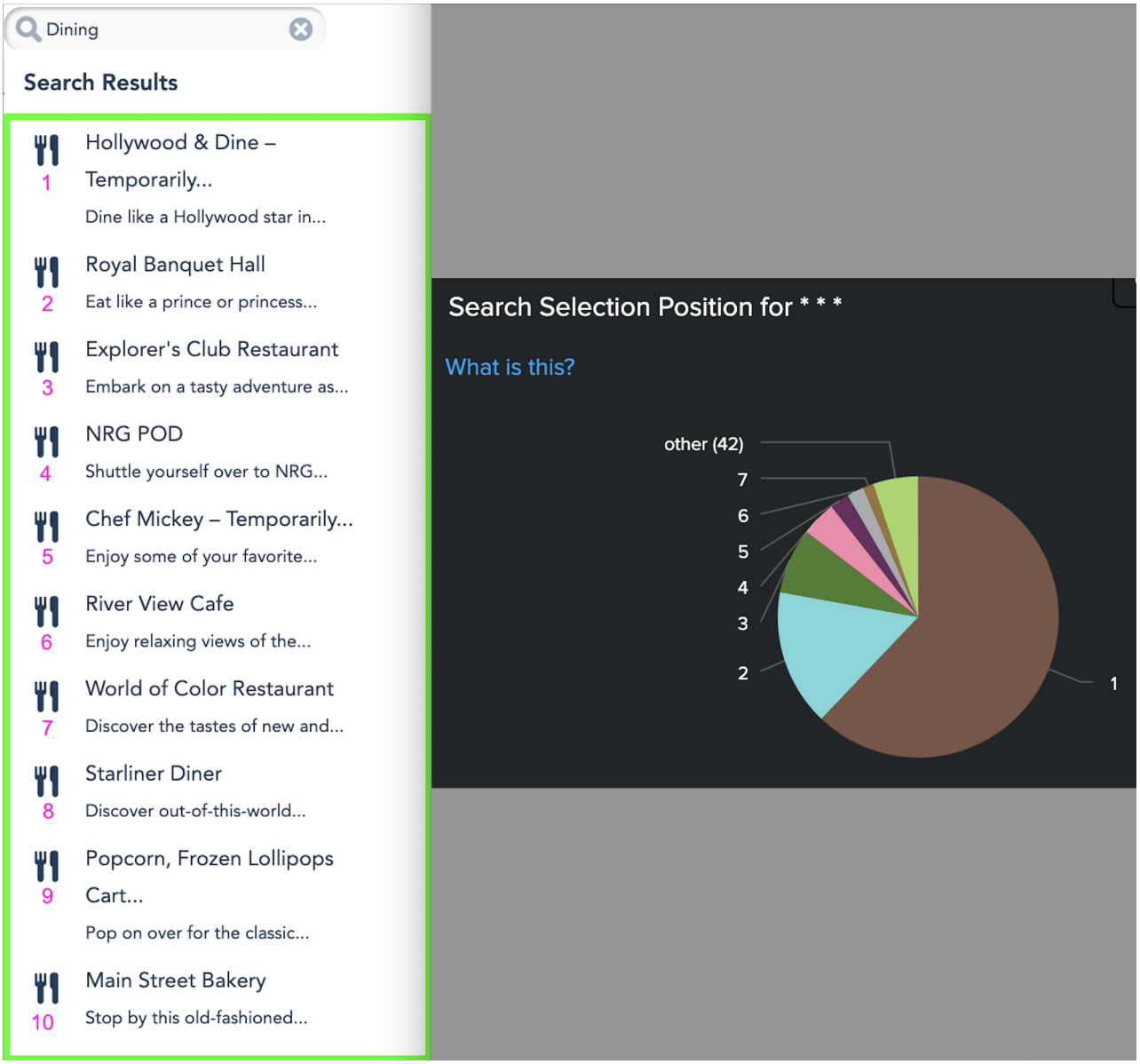

Figura 47: Selección de resultados registrada en un lapso de 60 minutos.

- Resultados más seleccionados:

Con todas las selecciones de búsqueda se creó el siguiente ranking, una especie de “Top 10”, con los resultados más seleccionados. Esto permite saber cuales son las tendencias dentro de la compañía, es decir cuales son las cosas más buscadas por los usuarios en un momento dado. Si por ejemplo, hay una nueva atracción, un nuevo evento el cual se está promocionando, o un restaurante que recién abre, este gráfico es un buen indicador para saber si los usuarios están buscando y están visitando dichas páginas, y en qué medida.

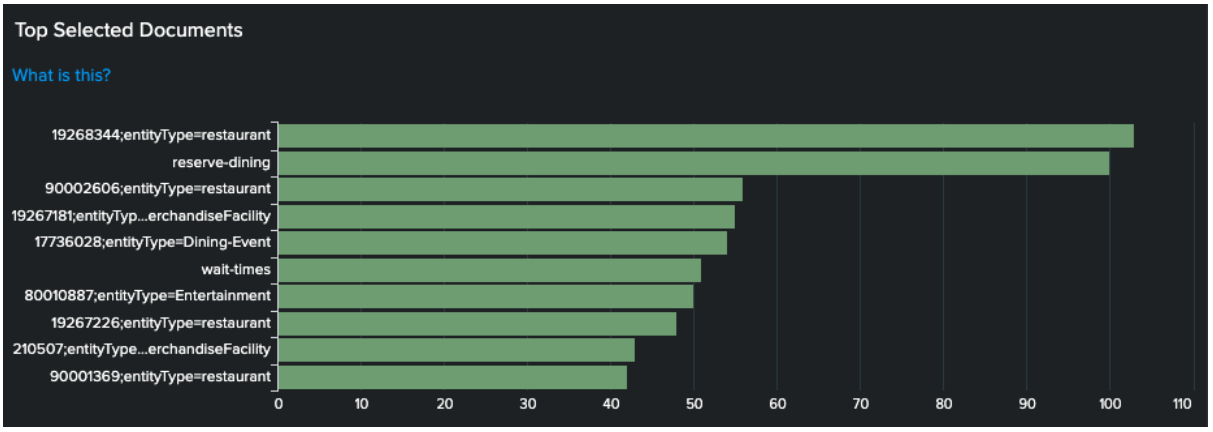

Figura 48: Ranking de documentos más seleccionados en un lapso de 60 minutos.

- Número de búsquedas por tipo:

La aplicación web no es la única que utiliza Search Service, hay otras aplicaciones de la empresa que también utilizan el servicio de búsqueda. Para saber el número de búsquedas que se reciben desde cada aplicación se creó el siguiente gráfico el cual las diferencia por su tipo. Los primeros dos tipos corresponden a la aplicación web legacy en la cual se realizó el trabajo de analytics. Lo que más utilizan los usuarios es la búsqueda desde la barra global de navegación. Se registraron un total de 67.329 búsquedas en tan solo 60 minutos, seguido por la búsqueda desde el listado principal con un total de 15.782 búsquedas realizadas. Este gráfico no se utiliza tanto como los otros, es solamente información que se recopila para tener estadísticas sobre su uso.

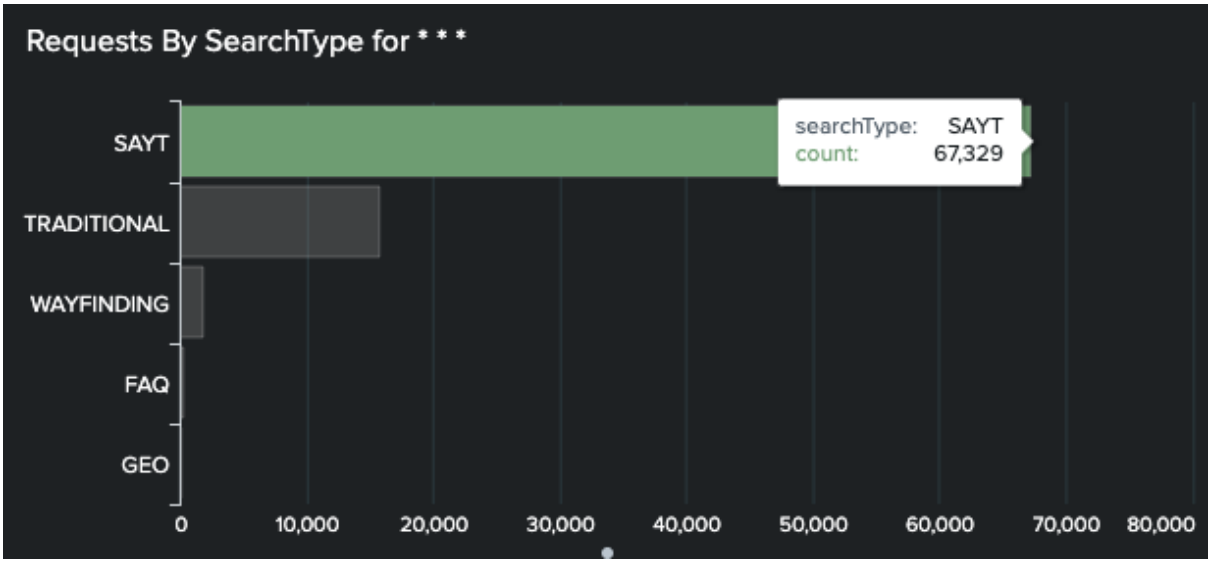

Figura 49: Número de búsquedas registradas provenientes de los inputs de búsqueda en un lapso de 60 minutos.

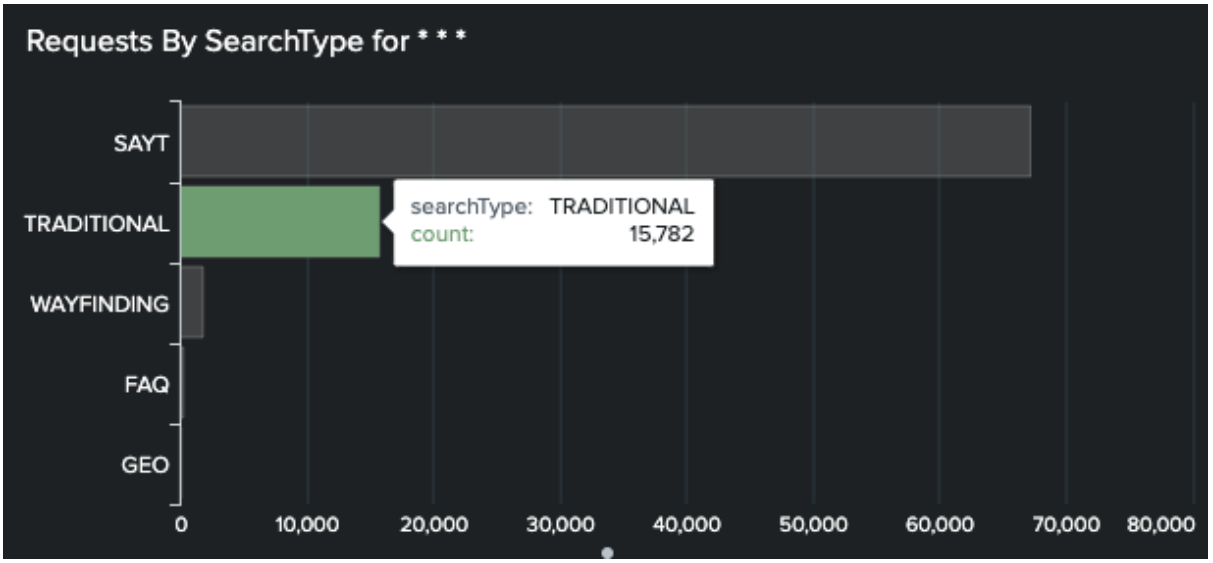

Figura 50: Número de búsquedas registradas provenientes del listado principal de búsqueda en un lapso de 60 minutos.

Capítulo 9

Conclusiones y trabajos futuros

La ejecución de este proyecto tuvo grandes beneficios para el cliente. Desde un punto de vista económico, y gracias a la abstracción del motor de búsquedas que se logró durante este trabajo, se pudo independizar y desacoplar dicha funcionalidad de la aplicación principal. Ahora cualquier actualización que se necesite del lado de Search Service puede llevarse a cabo sin la necesidad de tener que realizar una salida a producción de toda la aplicación web legacy. Como se mencionó en un principio las salidas a producción de dicha aplicación suelen ser muy costosas y requieren que varios equipos estén presentes: programadores de distintos equipos, testers, líderes, managers, productores de contenido, etc. Cada uno es responsable de una pequeña parte de la aplicación y tienen que estar presentes para validar y garantizar que todo sigue funcionando correctamente. A partir de esta implementación, y en caso de que se necesite actualizar algo (lógica, configuraciones, o lo que fuere dentro de Search Service), solo se necesita de un grupo reducido de personas, las cuales se encargan de mantener el servicio. Desconozco el monto que esto representa, ya que no tengo acceso a esa información, pero sí puedo decir que este trabajo hizo que dicho ahorro sea posible.

Ahora bien, desde un punto de vista tecnológico y funcional también se obtuvieron grandes mejoras. Sin dudas, uno de los aportes más importantes de esta Tesina es el trabajo de analytics que se llevó a cabo. Con dicho trabajo, se le pudo brindar al cliente toda la información relevante sobre las búsquedas que son realizadas en el sitio: desde que lugares se suelen hacer, qué suelen buscar los usuarios, así como también información sobre las selecciones que se realizan sobre los resultados. Toda esta información sirvió para que el equipo de Search Service pueda construir distintos gráficos. Gráficos relaciones al funcionamiento del servicio y también gráficos sobre el uso que le dan los usuarios. Gracias a la información proveída y a los gráficos construidos se pudieron detectar distintos tipos de problemas y plantear nuevas mejoras al servicio de búsqueda. La salida a producción de este trabajo fue totalmente transparente para el usuario, pero de gran importancia para la empresa.

Con respecto a los trabajos futuros, dado que la aplicación web legacy es realmente vieja, lo que se busca es que la misma quede obsoleta a mediano/largo plazo. Por lo que cada uno de los equipos deberá ir dejando la aplicación poco a poco e independizandose. De esta manera se tendrán varias aplicaciones pequeñas construidas a partir de nuevas tecnologías y no una tan grande y con tanta responsabilidad como la que se tiene ahora.

Actualmente estoy trabajando en una SPA la cual fue construida a partir de tecnologías nuevas como lo son Angular (versión 14) y Node.js. La misma ya se encarga de servir varias páginas que se migraron desde la aplicación web legacy,

incluyendo varias páginas de búsqueda. Se espera que la totalidad de las páginas de búsquedas sean servidas por esta nueva aplicación para que la aplicación web legacy quede deprecada finalmente. Es un gigante, el cual vamos desacoplando lentamente. Ya llevo casi 5 años trabajando en este proyecto y es realmente gratificante ver cómo un proyecto así poco a poco va llegando a su final.

Bibliografía

- Zend Framework: https://framework.zend.com/learn.html
- Elasticsearch: https://www.elastic.co/guide/index.html
- MVC pattern: https://developer.mozilla.org/en-US/docs/Glossary/MVC
- Object Inheritance PHP: https://www.php.net/manual/en/language.oop5.inheritance.php
- PHP exceptions: https://www.php.net/manual/en/language.exceptions.php
- PHP Error Object: https://www.php.net/manual/en/class.error.php
- jQuery Ajax requests: https://api.jquery.com/jquery.ajax/
- PHP Abstract Classes: https://www.php.net/manual/en/language.oop5.abstract.php
- Developer Tools: https://developer.chrome.com/docs/devtools/overview/