Universidad de Buenos Aires
Facultad de Ingeniería

75.61 - Taller de Programación III

Trabajo Práctico N° 3
RSVP

2º Cuat. - 2017

Titular ..……..	Andres Veiga
Jefe de Trabajos Prácticos....…..	Pablo Roca
Ayudante……...……..	Ezequiel Torres Feyuk
Alumno …….	Pablo Méndez

Fecha de re-entrega ..……..	16/10/2017

Índice

1. Introducción

El presente trabajo consta en desarrollar un sistema para confirmar invitaciones a eventos; mediante el cual, los usuarios podrán a un micrositio web en el que podrán completar un formulario con sus datos para reservar una vacante en un evento determinado, así como también se podrán realizar consultas sobre la asistencia de un determinado usuario a un evento y listar los usuarios que asistirán a un evento determinado.

Este sistema debe ser diseñado y desarrollado para correr bajo la plataforma de *Google Cloud*, haciendo uso del servicio gratuito de *Google App Engine*, el cual posee las siguientes características:

- Permite a empresas y otras organizaciones ejecutar sus aplicaciones web en la sólida infraestructura de Google.

- Proporciona un un entorno de aplicación completamente integrado, por lo que no requiere ensamblajes ni procesos complicados.

- Las aplicaciones pueden utilizar hasta 500 MB de almacenamiento permanente, y su ancho de banda alcanza a soportar 5 millones de visitas por mes.

- Incluye el soporte para tareas regulares, la importación y exportación de bases de datos, el acceso a datos protegidos con firewall, y soporte a estándares y a lenguajes como Python y Java, entre otras.

mediante los cuales se buscará cumplimentar los objetivos que este sistema implica y que se listan a continuación:

- Proveer un *front-end web* que le permita a los usuarios interactuar y utilizar las funcionalidades que el sistema ofrece a través de cada uno de sus módulos.

- Proveer un *back-end* desacoplado del *front-end* que brinde diferentes servicios al usuario con los siguientes objetivos:

    - Proveer un módulo de recepción y persistencia de confirmaciones a eventos.

    - Proveer un módulo de consultas por el **ID de confirmación** que permita conocer los datos de la persona que asistirá al evento.

    - Proveer un módulo de consultas por nombre, apellido, *email* ó compañía que permita conocer los usuarios que han confirmado la asistencia al evento.

    - Proveer un módulo de consultas para listar todas las confirmaciones recibidas por el sistema.

- Solo se podrá administrar un evento a la vez.

2. Solución propuesta

2.1. Arquitectura general

Uno de los puntos a analizar de la solución fue si la misma merecía ser desarrollada utilizando un estilo de arquitectura orientado a **Microservicios** ó utilizando un estilo tradicional de aplicación **3 - Tier Monolítica**; esto es, *Presentation Tier*, *Business Logic Tier* y *Database Tier*. El enfoque de microservicios; a diferencia del de n capas, requiere que tanto la capa de Lógica de Negocios como la de Base de Datos sean segmentadas y aisladas entre ellas verticalmente de acuerdo a una división lógica del modelo de negocios. Por lo que cada microservicio deberá ser autocontenido y no podrá acceder directamente a la capa datos de otros.

A continuación se muestra un diagrama comparativo de los dos estilos de arquitectura mencionados anteriormente.

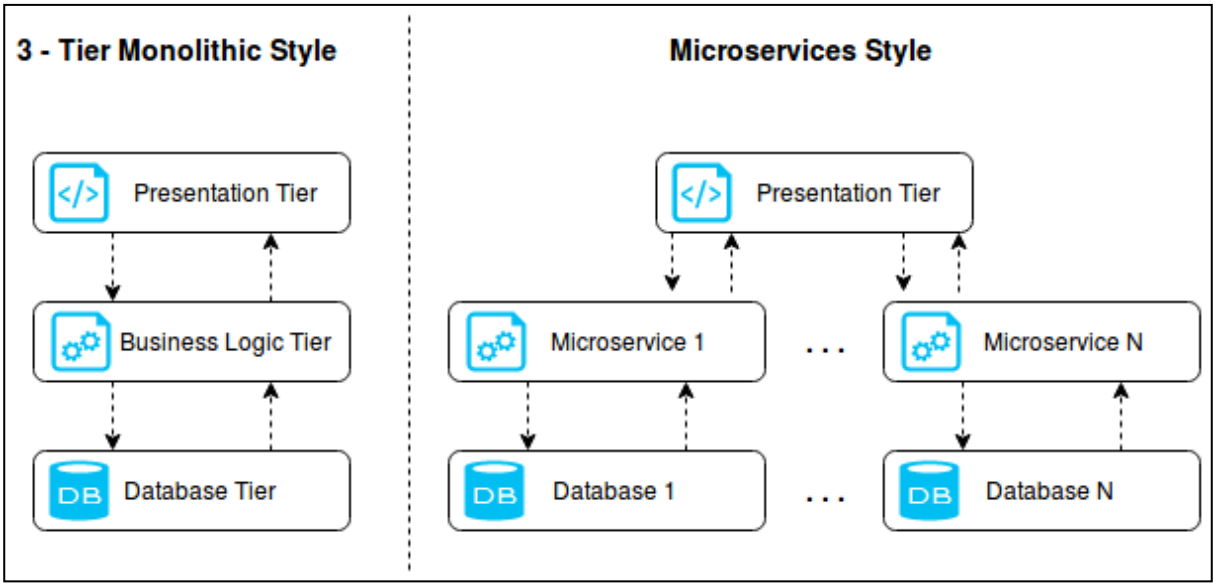

Figura 1. Estilos de arquitectura analizados

Para el caso de la aplicación a desarrollar; se optó por utilizar el primer estilo mencionado, debido a que si bien por la cantidad de funcionalidades a brindar sería posible dividir la lógica de negocios en varios microservicios, para todas ellas existe una sola entidad la cual representa a las personas que confirman su asistencia y que por la cantidad de atributos que la misma posee no resulta conveniente subdividirla para ser almacenada en varias bases de datos ya que esto complejiza el código que se debe desarrollar sobre los microservicios o la capa de aplicación para obtener toda la información necesaria de manera soportar la funcionalidad pedida.

A continuación se presenta un diagrama que muestra la arquitectura general de la solución desarrollada. Como puede observarse la capa de presentación corresponde a una **SPA** (*Simple Page Application*) desarrollada en **Angular JS 1.7** que es obtenida desde el *Front-End Service* desde **App Engine** y es ejecutada por el *Web Browser* del cliente. Esta página posee una dependencia con el *Back-End Service* quien mediante la exposición de diferentes *endpoints* provee la información y las funcionalidades necesarias que solicita el usuario final. Por otra parte la capa de la lógica de negocios, se encuentra implementada en lenguaje **GO** a través de otro servicio que se encuentra dentro del mismo proyecto que el Front-End y se comunica directamente con la cap de base de datos de invitados la cual se encuentra *hosteada* en un **Datastore** de **Google Cloud**.

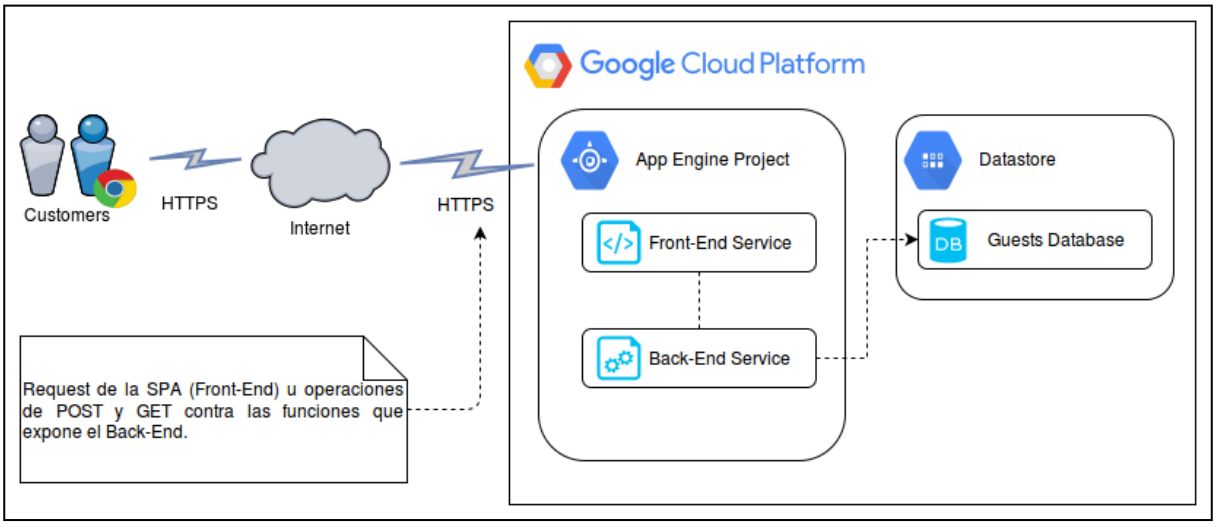

Figura 2. Arquitectura de la solución propuesta

2.2. Vistas de arquitectura 4+1

A continuación se presenta la solución propuesta basada en las vistas del modelo de arquitectura 4+1 de Kruchten. Para este caso se modelarán mediante sus respectivos diagramas las cuatro vistas (Lógica, Desarrollo, Procesos y Física) junto con los escenarios seleccionados, como se muestra a continuación.

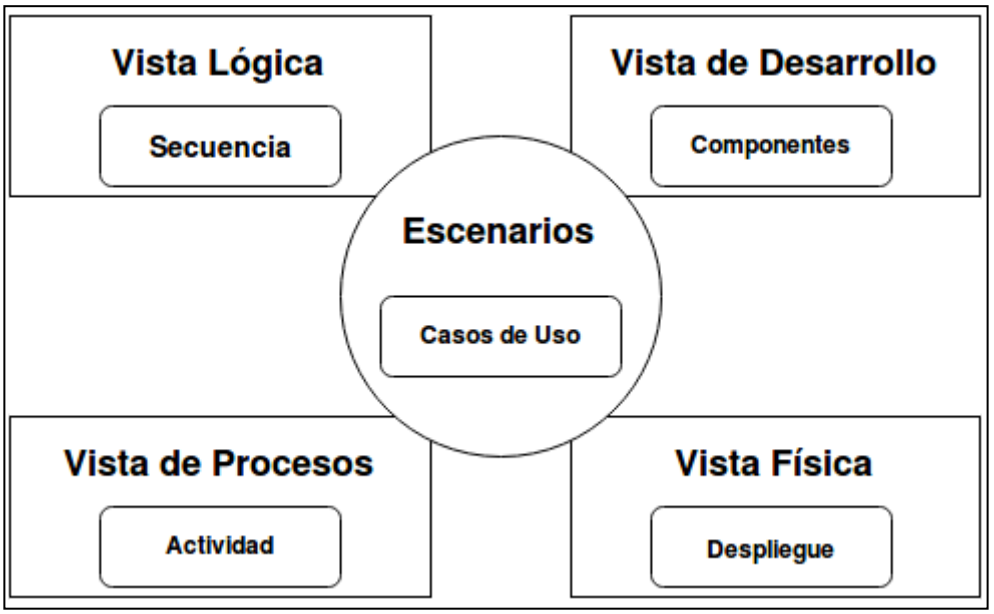

Figura 3. Diagrama de vistas de arquitectura del Modelo 4+1

- Vista Lógica: Esta vista presenta la funcionalidad que el sistema proporciona a los usuarios finales. Para su modelado se utilizarán los diagramas de clases.

- Vista de Desarrollo: Muestra cómo está dividido el sistema en componentes y las dependencias que hay entre ellos. Para su modelado se presentará un diagrama de componentes.

- Vista Física: Modela el sistema presentando un esquema de los componentes físicos, lógicos y como es la comunicación entre ellos. En este caso se utilizará un diagrama de despliegue.

- Vista de procesos: Muestra los procesos que conforman el negocio y como es la comunicación entre ellos. Para modelarla se utilizará un diagrama de actividades.

- Escenarios seleccionados: Esta vista es modelada por el diagrama de casos de uso y cumple la función de unir las otras cuatro vistas mediante la presentación de la interacción de los usuarios con el sistema.

2.1. Vista de escenarios seleccionados

A continuación se presenta el diagrama y las especificaciones de los casos de uso del sistema desarrollado.

2.1.1. Diagrama de Casos de Uso

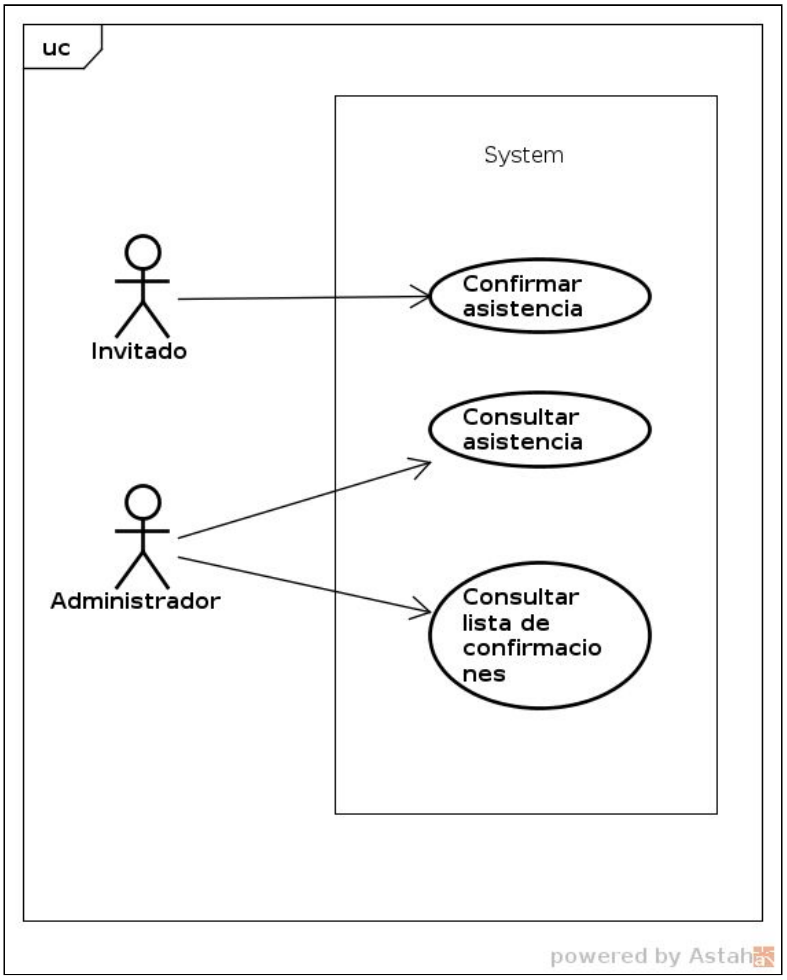

Figura 4. Diagrama de Casos de Uso

2.1.2. Especificación de Casos de Uso

Caso de Uso: CU001 - Confirmar Asistencia
Descripción: Permite a un invitado confirmar la asistencia al evento en curso.
Actores participantes: Invitado
Pre-condiciones: El evento debe estar disponible para confirmación.
Flujos

Flujo Principal
1 | El invitado accede al sistema para confirmar su asistencia al evento actual.
2 | El sistemas muestra un formulario en pantalla para ingresar los siguientes datos: Nombre, Apellido, Email, Compañía y el botón Confirmar asistencia.
3 | El invitado completa el Nombre, Apellido, Email, Compañía y presiona el botón Confirmar asistencia.
4 | El sistema procesa los datos ingresados y muestra en pantalla el siguiente mensaje: “La confirmación ha sido exitosa con el ID: {NÚMERO CONFIRMACIÓN}”; donde, NÚMERO CONFIRMACIÓN es el número asignado internamente por la aplicación para dicha operación.
Flujos Alternativos
-
Flujos de Excepción
-
Post-condiciones: La invitación del invitado queda confirmado en el sistema.

Caso de Uso: CU002 - Consultar Asistencia
Descripción: Permite al administrador del sistema consultar la asistencia de un invitado al evento en planificación.
Actores participantes: Administrador
Pre-condiciones: ● El evento debe estar disponible para consulta. ● El usuario que realiza la consulta debe estar dado de alta como administrador.
Flujos
Flujo Principal
1 | El administrador accede al sistema para realizar la consulta.
2 | El sistemas muestra la pantalla principal del sistema con las opciones **Consultar**

Asistencias y **Listar Invitados**.
3 | El administrador selecciona la opción Consultar Asistencias.
4 | El sistema muestra en pantalla un formulario con el campo **Ingrese el ID de la confirmación** y el botón **Consultar asistencia**.
5 | El administrador ingresa el ID de la confirmación a consultar y presiona el botón **Consultar asistencia**.
6 | El sistema procesa la consulta y muestra en pantalla el siguiente mensaje: “Asistencia confirmada del usuario = Nombre: {NOMBRE}, Apellido: {APELLIDO}, Email: {EMAIL}, Compañia: {COMPAÑIA}” {**E1**}
Flujos Alternativos
-
Flujos de Excepción
E1.1 | El sistema informa por pantalla “El ID ingresado es inválido”. Fin del Caso de Uso.
Post-condiciones: -

Caso de Uso: CU003 - Consultar lista de confirmaciones
Descripción: Permite al administrador del sistema listar los usuarios que han confirmado su invitación al evento.
Actores participantes: Administrador
Pre-condiciones: ● El evento debe estar disponible para consulta. ● El usuario que realiza la consulta debe estar dado de alta como administrador.
Flujos
Flujo Principal
1 | El administrador accede al sistema para realizar la consulta.
2 | El sistemas muestra la pantalla principal del sistema con las opciones **Consultar Asistencias** y **Listar Invitados**.

3 | El administrador selecciona la opción Listar Invitados.
4 | El sistema muestra en pantalla un filtro con los campos para completar **Usuario, Apellido, Email y Compañía** y el botón **Buscar**.
5 | El administrador ingresa el ID de la confirmación a consultar y presiona el botón **Consultar asistencia**.
6 | El sistema procesa la consulta y muestra en pantalla la lista de invitados encontrados con el siguiente formato: **ID | Usuario | Apellido | Email | Apellido** y los botones **Next** y **Prev** para poder navegar entre las páginas de resultados.
Flujos Alternativos
-
Flujos de Excepción
-
Post-condiciones: -

2.2. Vista de procesos

A continuación se presentan los diagramas de actividades modelados para representar el negocio sobre el cual está basado el sistema.

Escenario: Confirmación de una asistencia

Descripción: Flujo de actividades donde el invitado ingresa al micrositio del evento y confirma su asistencia enviando sus datos.

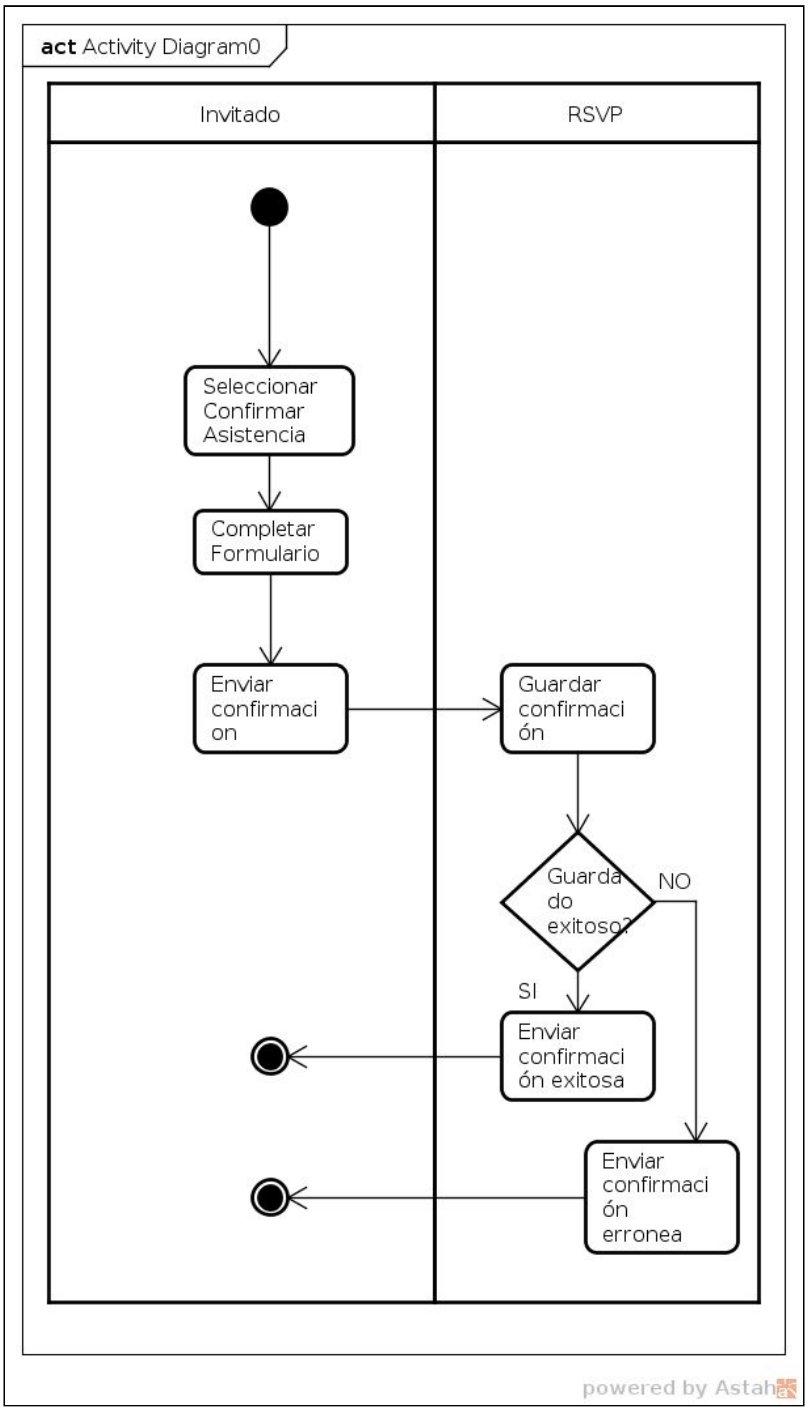

Figura 5. Diagrama de Actividades de la confirmación de una asistencia

Escenario: Consulta de invitados confirmados

Descripción: Flujo de actividades donde se muestran las operaciones que puede realizar una administrador del sistema. La primera es buscar un invitado por su ID de confirmación y la segunda es listar los invitados de acuerdo a algún filtro de búsqueda.

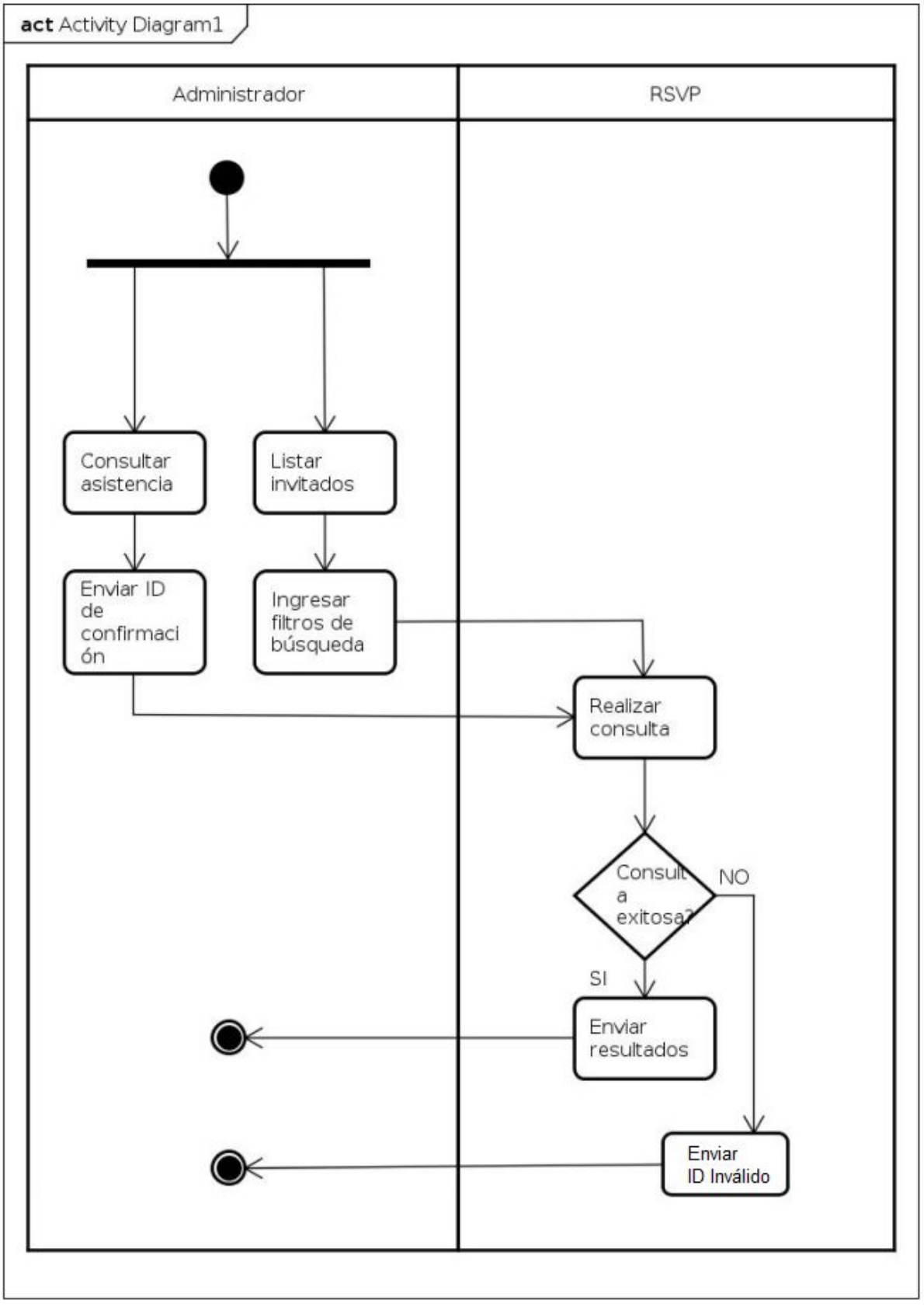

Figura 6. Diagrama de Actividades de la consulta de una asistencia

2.3. Vista de Desarrollo

A continuación se presenta el diagrama de componentes del sistema. Como se puede observar el mismo posee cuatro componentes fundamentales dos de los cuales fueron desarrollados para el presente trabajo (*Front-End* y *Back-End*), mientra que los otros dos (*Datastore* y *Memcache*) son componentes proveídos por la plataforma de *App Engine*.

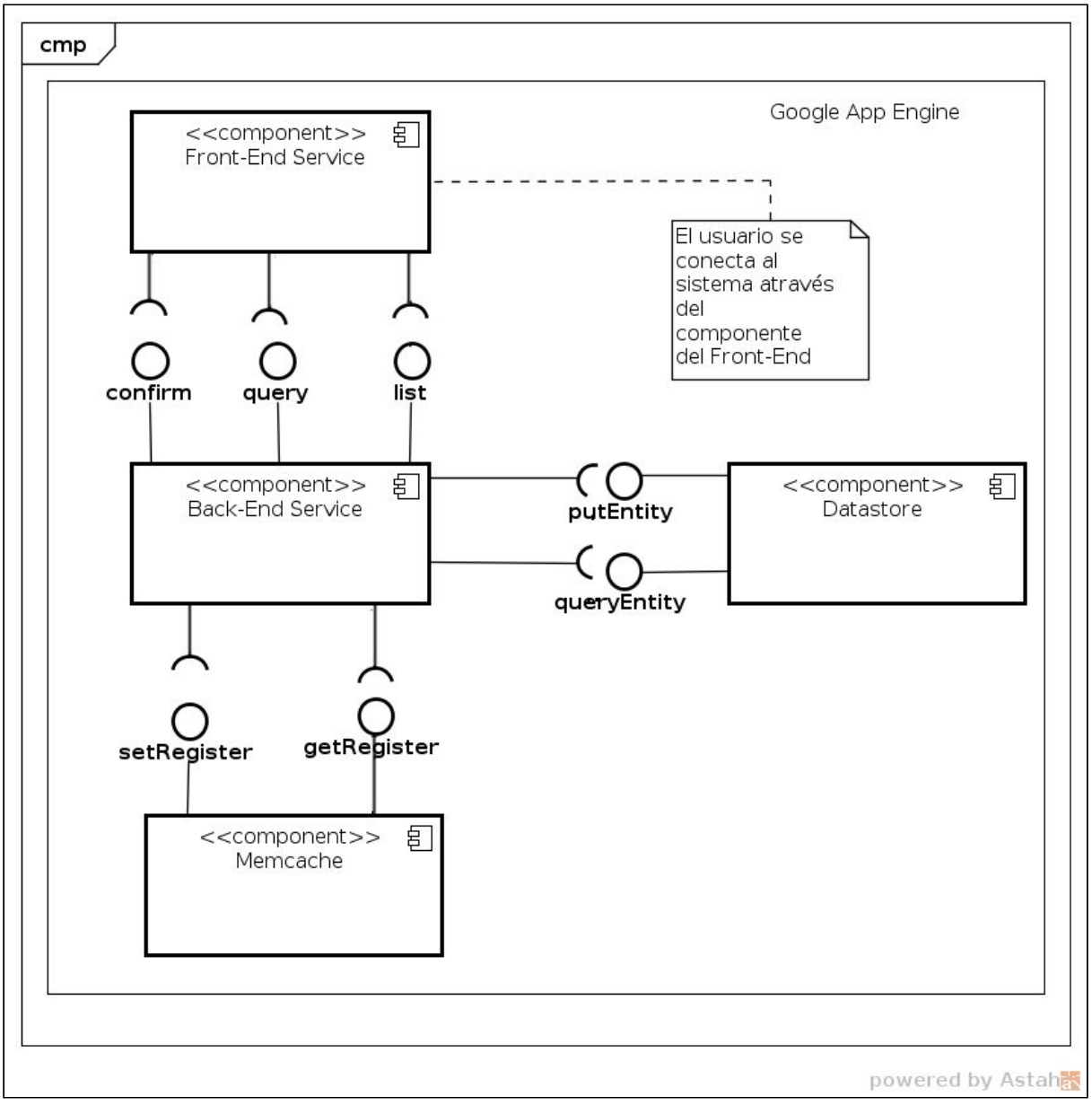

Figura 7. Diagrama de Componentes del sistema

A continuación se detalla brevemente el propósito de cada uno de ellos:

Back-End: Como se puede observar el componente central corresponde con el Back-End quien posee la lógica de negocios de la aplicación y por lo cual consume y provee información a los otros tres mediante un juego de interfaces definidas.

Front-End: Es el componente encargado de proporcionar una interfaz de acceso al usuarios final y proporcionarle la navegabilidad necesaria para moverse dentro del sistema.

Memcache: Es un servicio de *cache* de datos proporcionado por la plataforma de Google que en la aplicación es utilizado para almacenar los resultados de las búsquedas recientes de manera que en las sucesivas consultas exista la posibilidad que que los datos se encuentren en esta última y así evitar la búsqueda sobre el *Datastore* y enviar la respuesta más rápidamente.

Datastore: Es un servicio de almacenamiento permanente de datos proporcionado por la plataforma de Google el cual es utilizado para persistir la información proporcionada por los invitados al evento.

2.4. Vista de Física

A continuación se presenta el diagrama de despliegue con la distribución de los componentes. Dichos componentes pueden ser nodos (servidores) o artefactos (aplicaciones o bases de datos), los cuales se describen a continuación:

- **FrontEndServer**: Nodo que puede tener varias instancias del artefacto *Front-End* de manera de poder atender los pedidos de la página web por parte de los usuarios.

- **BackEndServer**: Nodo en donde puede haber corriendo varias instancias del *Back-End* de manera de poder atender los request de confirmación y consulta de invitados al evento.

- **StorageServer**: Nodo en el que se encuentra corriendo el datastore donde se almacenan las entidades con la información enviada por los asistentes al evento.

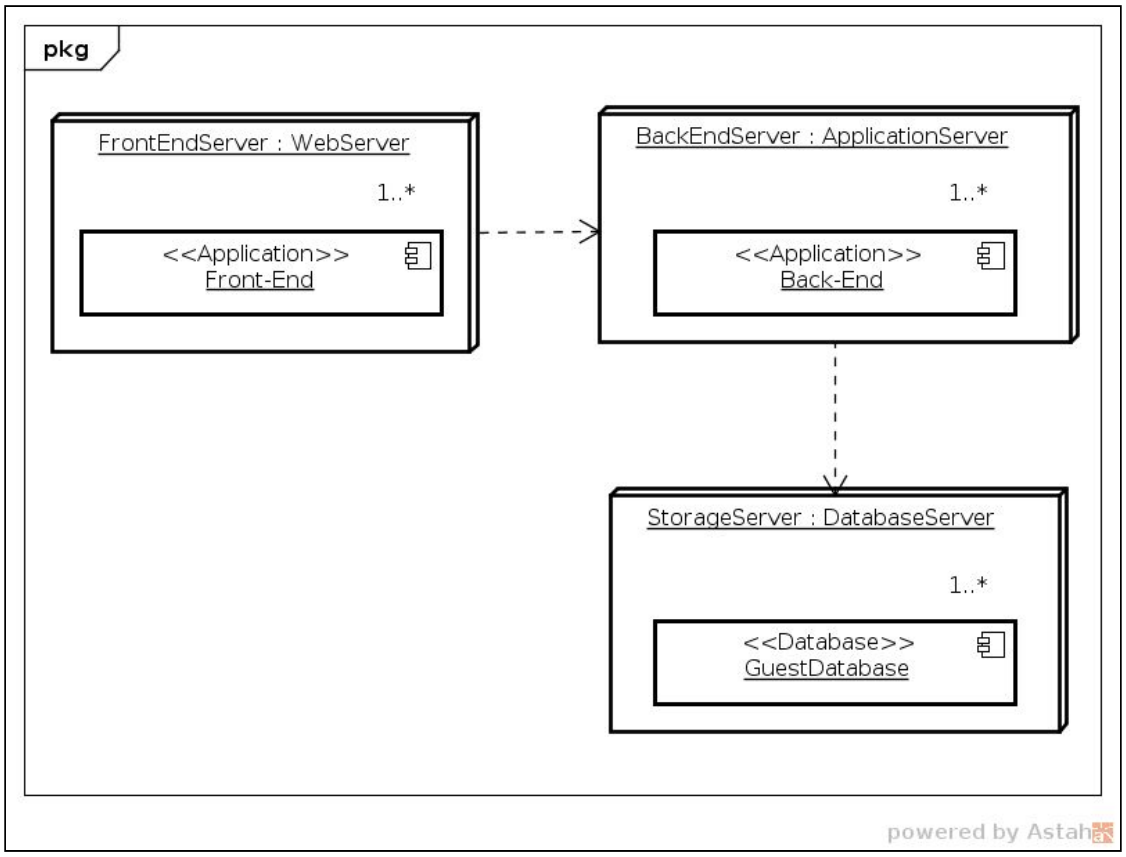

Figura 8. Diagrama de Despliegue del sistema

3. Desglose de actividades

A continuación se presenta el desglose de tareas planificadas para realizar el trabajo. En la siguiente tabla se lista la secuencia de tareas, el tiempo estimado en horas así como también el tiempo real de las mismas.

Tarea	Fecha Estimada	Horas Estimadas	Fecha Real	Horas Reales
Investigación de implementación de microservicios en App Engine	15/09/2017 - 16/09/2017	2	16/09/2017 - 16/09/2017	4
Implementación de microservicio de prueba en GO	16/09/2017 - 17/09/2017	2	16/09/2017 - 17/09/2017	8
Análisis y planeamiento del TP	19/09/2017 - 20/09/2017	2	18/09/2017 - 18/09/2017	2
Implementación y prueba de la aplicación	21/09/2017 - 24/09/2017	10	19/09/2017 - 24/09/2017	20
Test de Carga	24/09/2017 -	4	26/09/2017 -	8

	25/09/2017		27/09/2017	
Preparación del Informe	26/09/2017 - 26/09/2017	4	24/09/2017 - 27/09/2017	8
Preparación de la Demo	27/09/2017 - 27/09/2017	4	27/09/2017 - 27/09/2017	8
Total	-	28	-	58

Tabla 1. Estimación de tiempos

Como puede observarse en la Tabla 1, la diferencia entre el tiempo total estimado y el tiempo real utilizado para el desarrollo del trabajo es de 30 horas. Analizando cada una de las tareas, se advierte que si bien casi todas fueron subestimadas, aquellas cuya estimación fue más grosera fueron las de “Implementación de microservicio de prueba en GO” e “Implementación y prueba de la aplicación”. Esto es debido a que a la hora de hacer los cálculos no se tuvo en cuenta que el estudio de un nuevo estilo de arquitectura y su implementación en un lenguaje desconocido; además de tener que realizarlo en una plataforma en la nube de Google donde la documentación es muy difícil de encontrar, leer y entender, lleva mucho más tiempo y esfuerzo que en otro caso.

Para el caso de la tarea de “Test de carga” sucedió algo parecido, ya que no se tuvo en cuenta que por empezar este tipo de pruebas requieren mucho tiempo y además se prestó la necesidad de aprender a utilizar una nueva herramienta nunca antes usada (JMeter), lo cual atraso aún más los tiempos.

Por último; para el resto de las tareas subestimadas, las mismas ya fueron realizadas para otros trabajos. Simplemente se debería haber agregado algunas horas más de contingencia a las calculadas para evitar el desfase de las mismas. Este último criterio también se debería haber aplicado al resto de las tareas.

4. Pruebas de carga

A continuación se presentan las pruebas realizadas al sistema subido a la nube de Google. Las mismas fueron realizadas con la aplicación Apache JMeter V3.2. Esta herramienta herramienta open source implementada en Java permite realizar test de comportamiento funcional y medir el rendimiento de las aplicaciones. También se puede utilizar para realizar pruebas de stress y carga; por ejemplo, en un servidor, y poner a prueba su rendimiento. También se utilizó el plugin *Throughput Shaping Timer* para *JMeter* de manera de poder controlar la cantidad de request enviados al *App Engine* durante las pruebas.

Las pruebas de carga se han efectuado sobre ambos servicios desarrollados:

- **Front-End**: https://useful-music-180113.appspot.com
- **Back-End**: https://app-dot-useful-music-180113.appspot.com

Para ello se han creado y configurado varios casos de prueba con el fin de medir los límites de los dos componentes mencionados anteriormente:

4.1. Prueba N° 1: Test de carga sobre el Front-End

En esta prueba se pretende realizar *requests* sobre el *Front-End* de la aplicación. Como el mismo fue implementado como SPA, la prueba se realizará mediante peticiones GET las cuales descargarán completamente el sitio web a la pc del usuario. El objetivo de este *test* es poder determinar la cantidad máxima de usuarios que podrían estar ingresando al sitio sin experimentar errores o retrasos en su descarga.

Ejecución de la prueba

Para la misma se configuró el patrón mostrado en la Figura 9. En la misma se puede ver que se inicia con 100 requests por segundo, luego se aumenta a 300, luego a 700 y 1000 y a partir de allí se aumenta de a 500 hasta llegar a 3000. Todos con un intervalo de 10 minutos de diferencia.

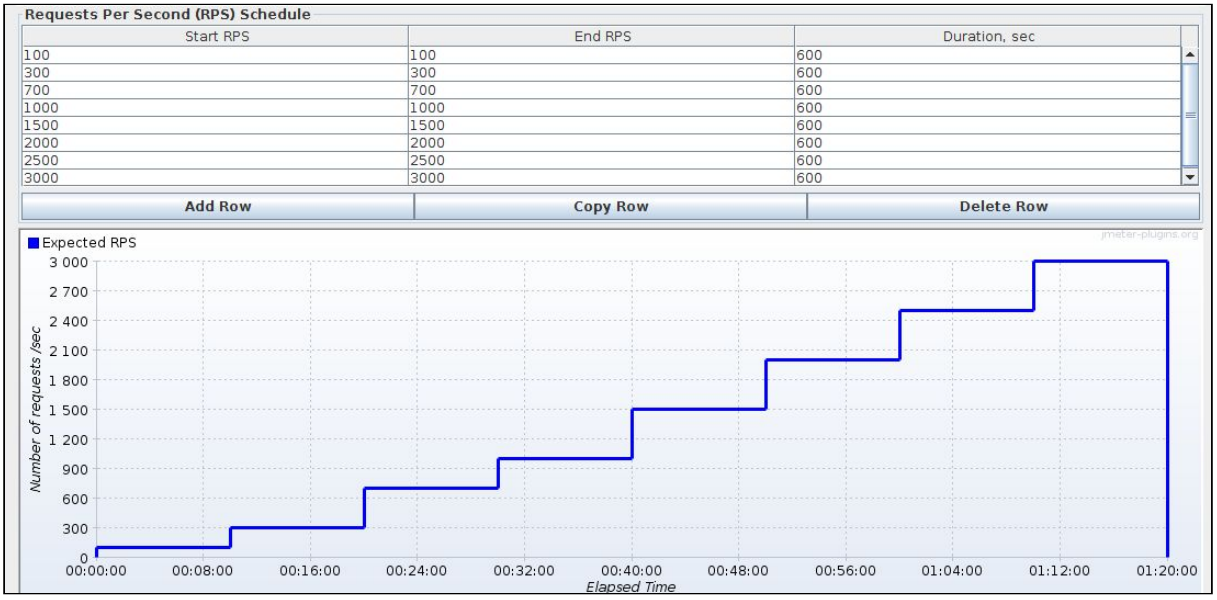

Figura 9. Configuración de caso de prueba del Frontend en JMeter

Resultados de la prueba

Observando los resultados obtenidos en JMeter mostrados en la Tabla 2 y los perfiles de transacciones por segundo ejecutadas en la prueba, se puede observar que los *requests* exitosos se mantienen uniformes a lo largo de toda la prueba y solamente se producen picos de fallas cada 10 minutos que es justo cuando se aumenta el escalón para luego estabilizarse. Estos picos de error son cada vez más grandes conforme se encuentra más cargado el sistema.

Requests	Executions			Response Times (ms)							Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	90th pct	95th pct	99th pct	Throughput	Received	Sent
Total	349469	165693	47.41%	13297.52	1	4158796	17559.90	129357.75	484404.43	69.74	153.61	4.84
HTTP Request	349469	165693	47.41%	13297.52	1	4158796	17559.90	129357.75	484404.43	69.74	153.61	4.84

Tabla 2. Resumen de JMeter de la prueba del Frontend

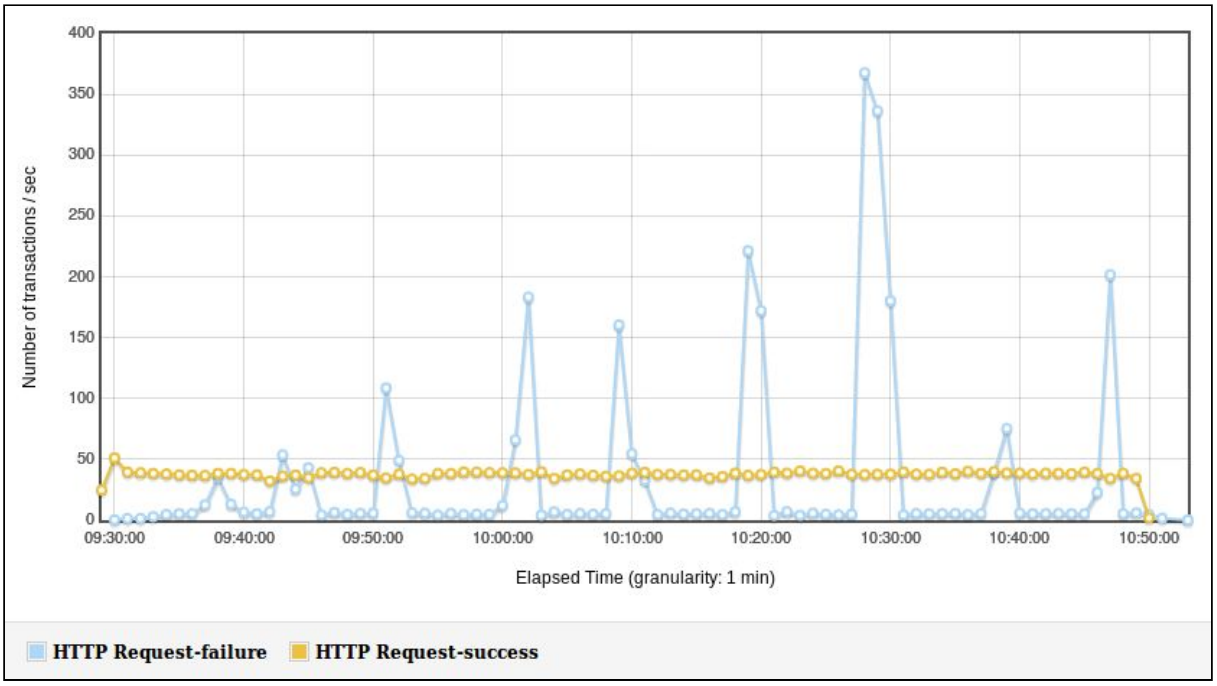

Figura 10. Resumen de JMeter de la prueba del Frontend

Contrastando estos resultados con los gráficos de App Engine (Figura 11) se puede observar un patrón similar en las solicitudes recibidas llegando a un máximo de 50 requests por segundo. Estos picos pueden ser evitados realizando un salto más suave entre los peldaños del escalón o bien realizando subidas de menos cantidad de *requests*. Más allá de esto se puede ver que el sistema se repone del transitorio y rápidamente comienza a responder la totalidad de los requerimientos de manera exitosa.

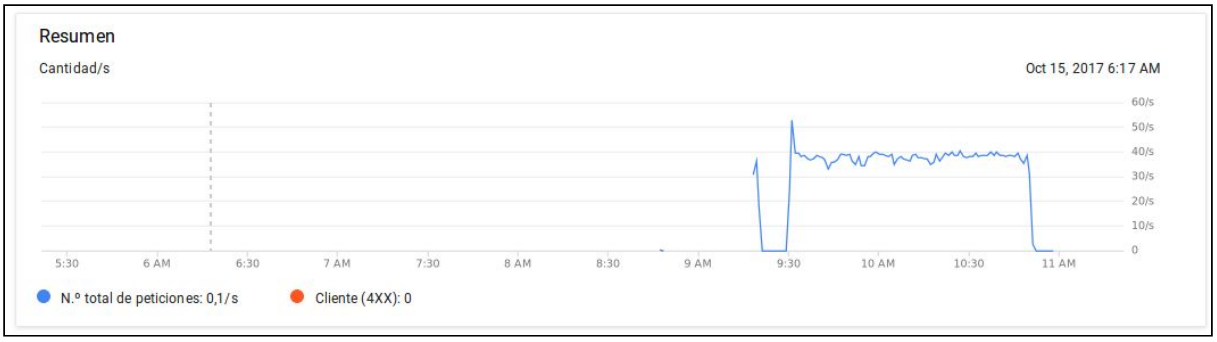

Figura 11. Perfil de solicitudes recibidas en App Engine

Un caso interesante para analizar es el de la cantidad de instancias que tuvo que levantar App Engine para esta prueba. Como puede observarse en al Figura 12, no se levantó ninguna instancia del servicio y esto es debido a como trabaja y cómo está

estructurada la plataforma (Figura 13). Al estar el *Frontend* configurado para servirse como contenido estático, los *requests* que llegan son interceptados por el **Google Front End** y son redireccionados a la **Edge Cache** donde automáticamente son respondidos con una copia de la **SPA**. Este comportamiento es deseable y recomendado por Google en la documentación de App Engine ya que de esta manera se le quita carga de procesamiento al **App Engine Frontend** y no es necesario que el mismo tenga que levantar instancias de procesamiento.

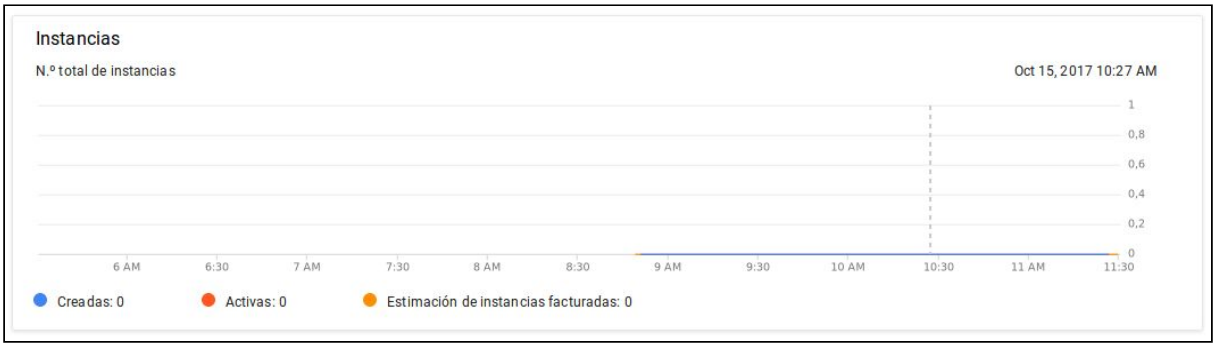

Figura 12. Instancias levantadas durante la prueba del Frontend

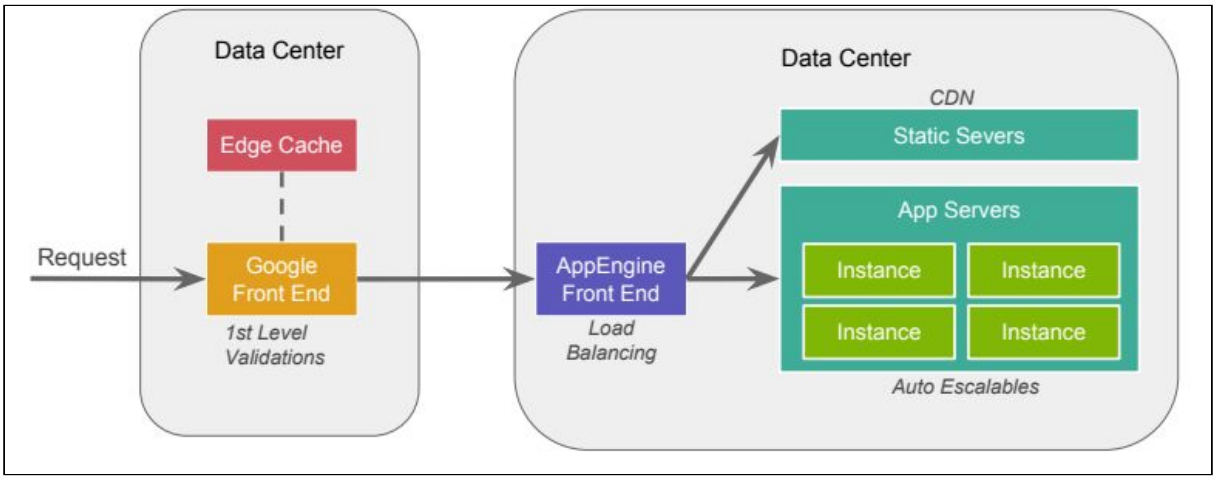

Figura 13. Arquitectura de App Engine

4.2. Prueba N° 2: Test de carga sobre el Back-End

En esta prueba se confirmaciones contra el servicio de *Back-End* de la aplicación. Como este servicio requiere procesamiento y acceso al Datastore por parte de la plataforma de App Engine y teniendo en cuenta el resultado anterior, es de esperar que la cantidad de usuario que pueda soportar operando al mismo tiempo sin generar muchos errores sea menor.

Ejecución de la prueba

De antemano se cargó el Datastore con información (Figura 14) de manera de poder probar el procesamiento en las consultas y además la utilización de la *Memcache*. Es por eso que se cargó un caso de prueba en JMeter con el mismo perfil de solicitudes mostrado

en la Figura 9 y un conjunto de 600 entidades distintas a consultar, obteniendo los resultados mostrados a continuación.

Figura 14. Datastore cargado

Resultados de la prueba

Como puede observarse (Figuras 16 y 17) los perfiles de requerimientos hechos y recibidos desde JMeter y App Engine son bastante parecidos y en el caso del gráfico de JMeter se puede observar que llegando al final de la prueba; donde aumenta bastante la cantidad de requerimientos por segundo realizados, comienza a crecer la cantidad de respuestas fallidas lo que hace que el porcentaje global de error (Figura 15) aumente llegando de esta manera a los límites del sistema con 3000 *request* por segundo.

Requests	Executions			Response Times (ms)							Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	90th pct	95th pct	99th pct	Throughput	Received	Sent
Total	**349469**	**165693**	**47.41%**	**13297.52**	**1**	**4158796**	**17559.90**	**129357.75**	**484404.43**	**69.74**	**153.61**	**4.84**
HTTP Request	349469	165693	47.41%	13297.52	1	4158796	17559.90	129357.75	484404.43	69.74	153.61	4.84

Figura 15. Resumen de JMeter de la prueba del Backend

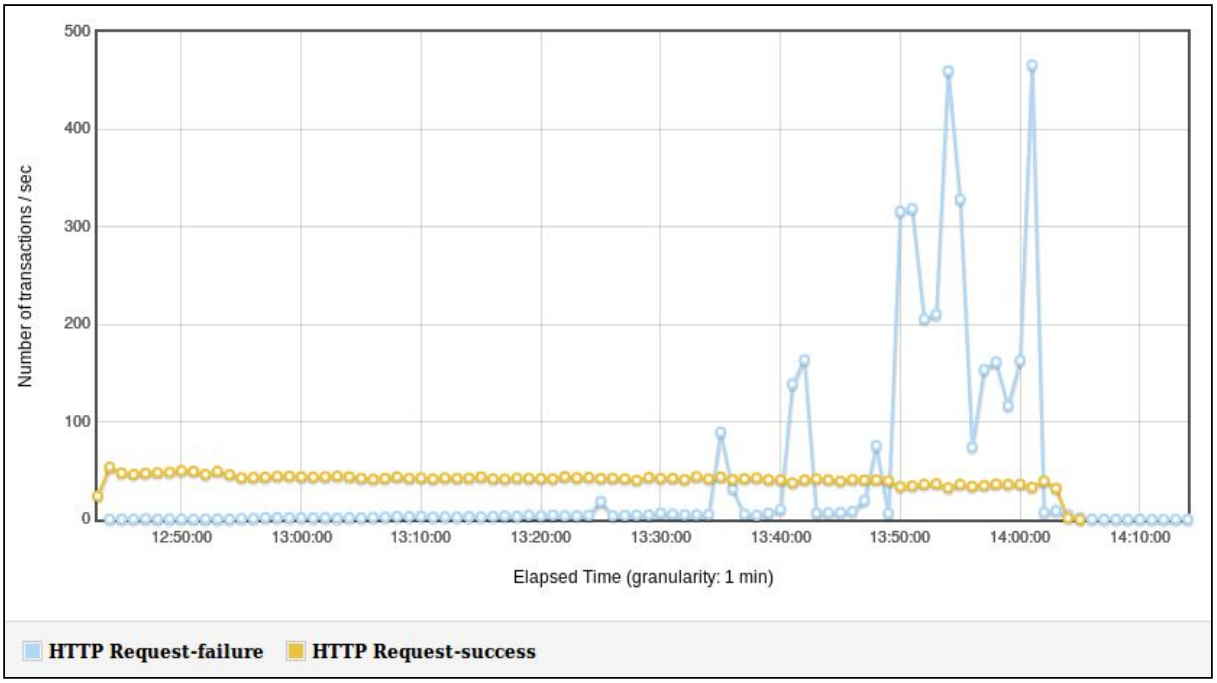

Figura 16. Perfil de solicitudes obtenido en JMeter

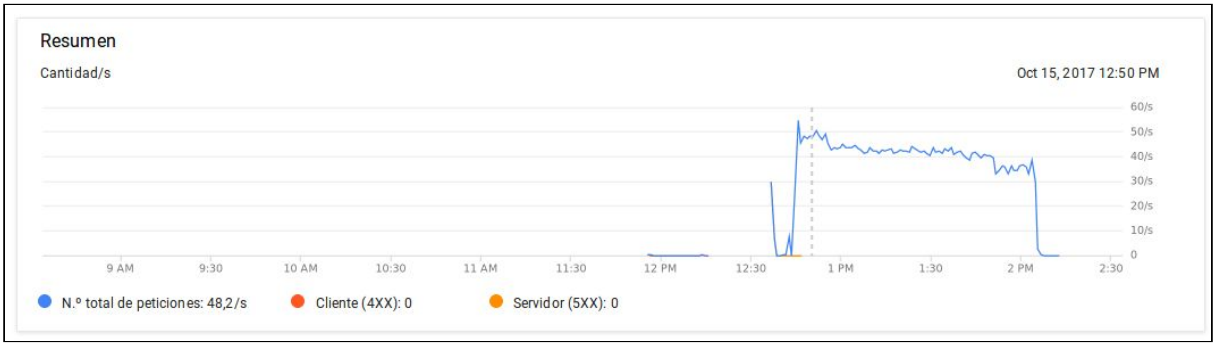

Figura 17. Perfil de solicitudes recibidas en App Engine

Un caso interesante de observar en esta prueba es la utilización de la Memcache (Figura 18) donde se puede ver que se hace uso intensivo de la misma mejorando el throughput total del sistema y pudiendo responder más rápidamente los pedidos de los usuarios.

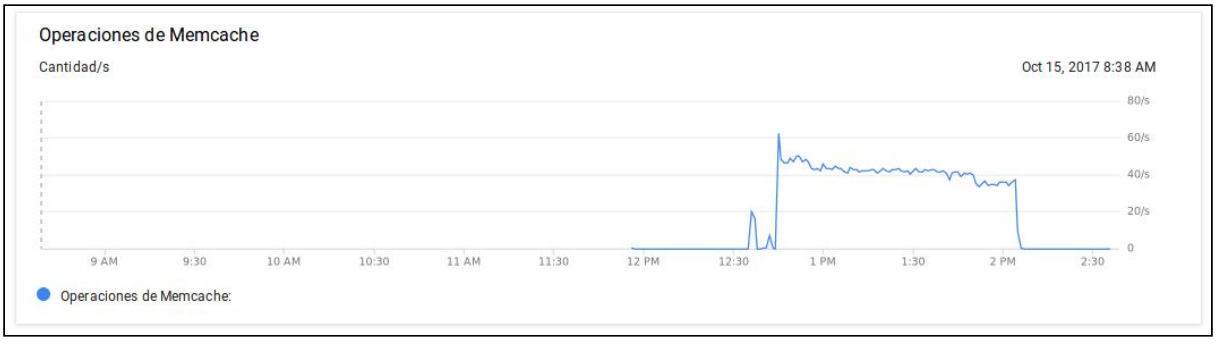

Figura 18. Operaciones en la Memcache

Por último se puede observar que para responder a las peticiones de el caso de prueba, la plataforma tuvo que levantar 3 instancias del Backend (Figura 19).

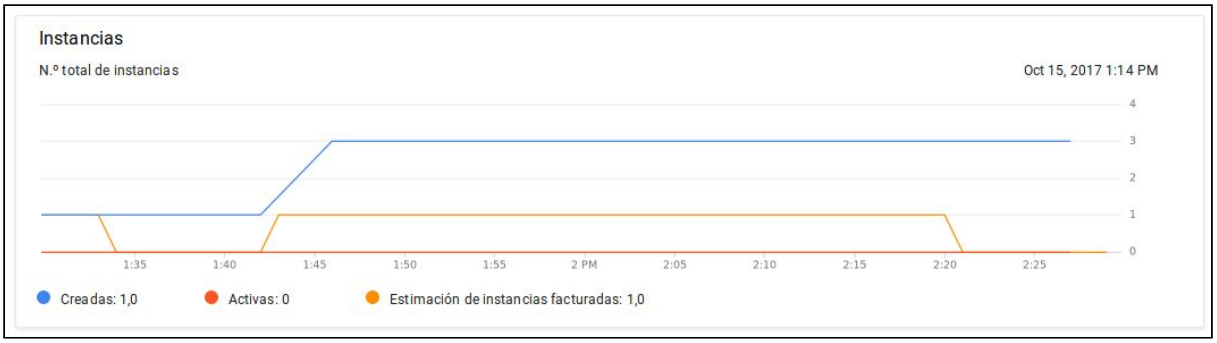

Figura 19. Instancias del Backend levantadas

Al igual que en el caso de prueba anterior se podría probar la carga del sistema con escalones con pendientes más suaves ó saltos menos abruptos de manera de ver si se logran levantar más instancias del *Backend*.

5. Código fuente

A continuación se presenta el código fuente de los componentes desarrollados para implementar la solución descrita anteriormente.