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 entrega …….. 28/09/2017

# 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 responsive* 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.

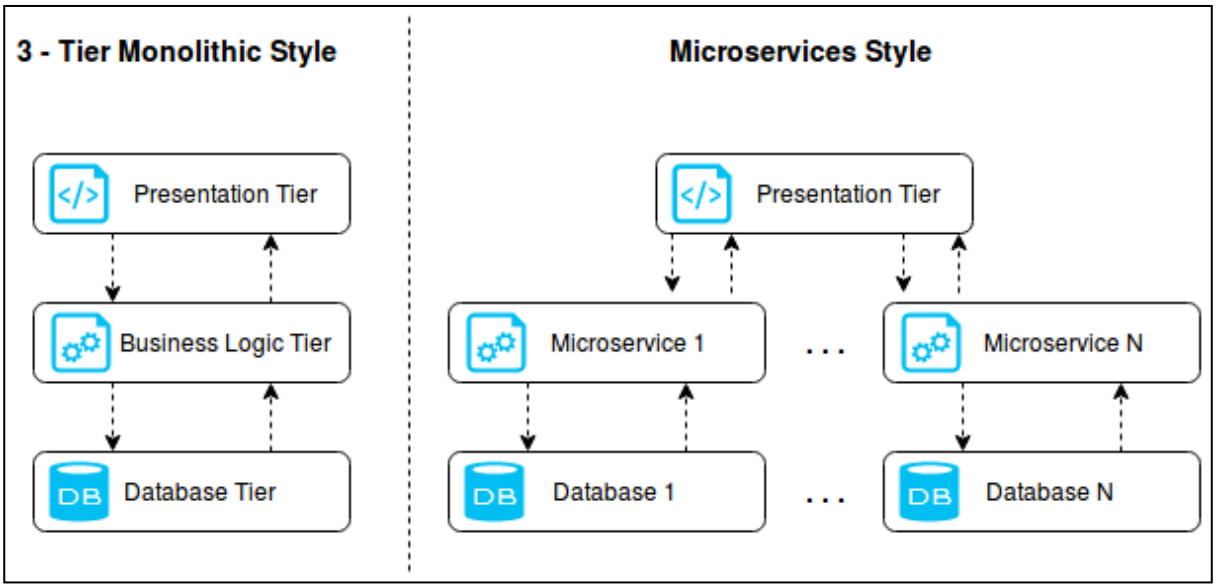
- Solo se podrá administrar un evento a la vez.

# 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.

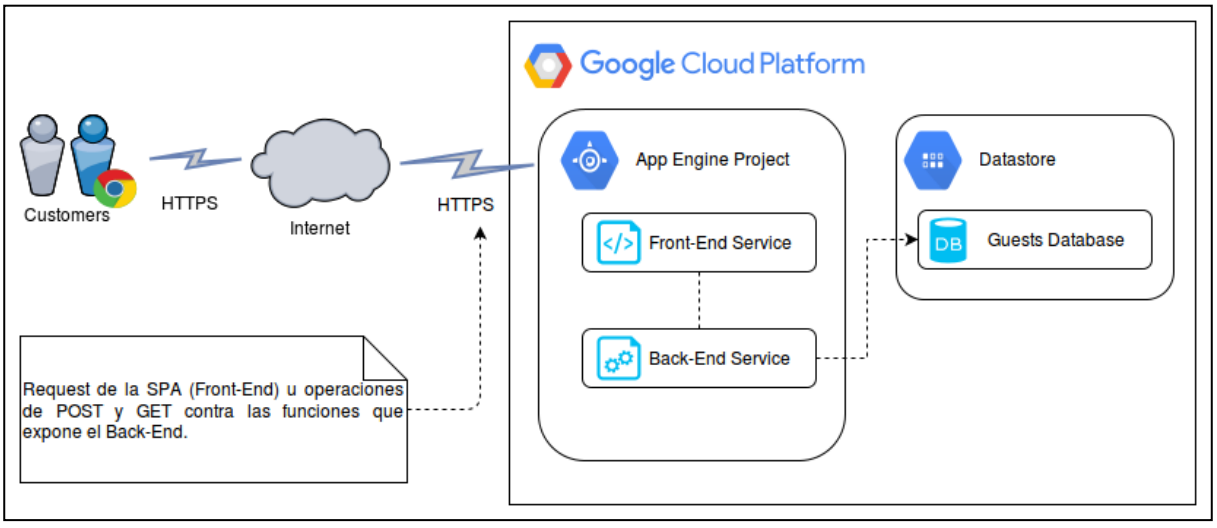
A continuación se muestra un diagrama comparativo de los dos estilos de arquitectura mencionados anteriormente.




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

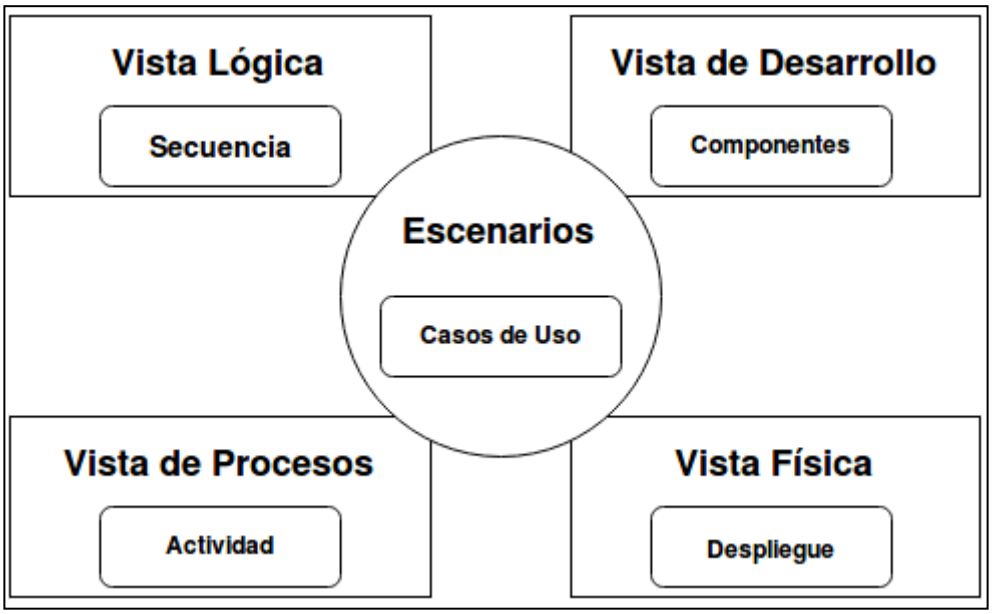
microservicios o la capa de aplicación para obtener toda la información necesaria de manera soportar la funcionalidad pedida.

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**.




## 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.




- 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.

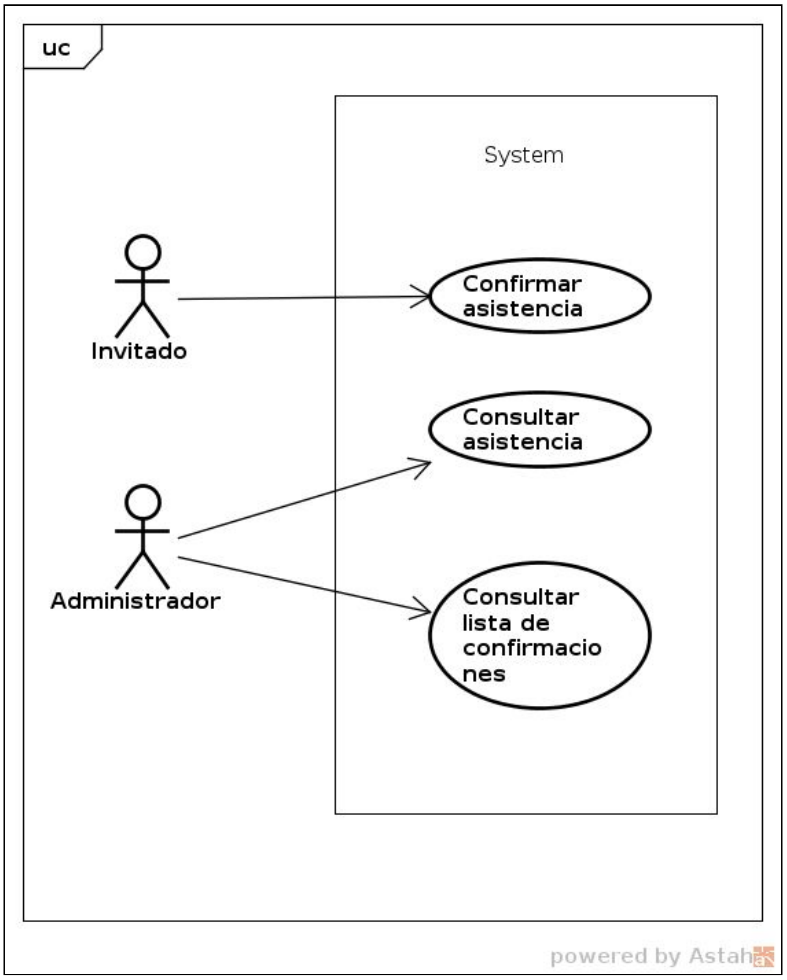
- Vista de procesos: Muestra los procesos que conforman el negocio y como es la comunicación entre ellos. Para modelarla se utilizará un diagrama de actividades.

- 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




### 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**: Invitado

**Precondiciones**:

- El evento debe estar disponible para confirmación.

**Restricciones**: El invitado solo podrá confirmar su asistencia al único evento que se encuentra actualmente en planificación.

**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 (**E1 Datos Inválidos**) (**E2 Error Inesperado**).

**Flujos de Excepción**

**E1 Datos Inválidos**

1. El sistema informa por pantalla “Los datos ingresados son inválidos”.
2. El sistema retorna al punto 2 del flujo principal.

**E2 Error Inesperado**

1. El sistema informa por pantalla “Ha ocurrido un error inesperado”.
2. El sistema retorna al punto 2 del flujo principal.

**Caso de uso**: CU002 - Consultar asistencia

**Descripción**: Permite al administrador del sistema consultar la asistencia de un invitado al evento en planificación.

**Actores**: Invitado

**Precondiciones**:
- El evento debe estar disponible para consulta.
- El usuario que realiza la consulta debe estar dado de alta como administrador.

**Restricciones**: -

**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 Datos Inválidos**) (**E2 Error Inesperado**) (**E3 Usuario no encontrado**).

**Flujos de Excepción**

**E1 Datos Inválidos**

1. El sistema informa por pantalla “Los datos ingresados son inválidos”.
2. El sistema retorna al punto 2 del flujo principal.

**E2 Error Inesperado**

1. El sistema informa por pantalla “Ha ocurrido un error inesperado”.
2. El sistema retorna al punto 2 del flujo principal.

**E3 Usuario no encontrado**

1. El sistema informa por pantalla “No existe ninguna confirmación para el ID: {ID}”.
2. El sistema retorna al punto 2 del flujo principal.

**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**: Administrador

**Precondiciones**:
- El evento debe estar disponible para consulta.
- El usuario que realiza la consulta debe estar dado de alta como administrador.

**Restricciones**: -

**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.
7. El sistema muestra en pantalla un filtro con los campos para completar **Usuario, Apellido, Email y Compañía** y el botón **Buscar**.
8. El administrador completa los datos que desea del filtro y presiona el botón **Buscar**.
9. 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(**E1 Datos Inválidos**) (**E2 Error Inesperado**) (**E3 Usuarios no encontrados**).

**Flujos de Excepción**

**E1 Datos Inválidos**

1. El sistema informa por pantalla “Los datos ingresados son inválidos”.
2. El sistema retorna al punto 2 del flujo principal.

**E2 Error Inesperado**

1. El sistema informa por pantalla “Ha ocurrido un error inesperado”.
2. El sistema retorna al punto 2 del flujo principal.

**E3 Usuarios no encontrados**

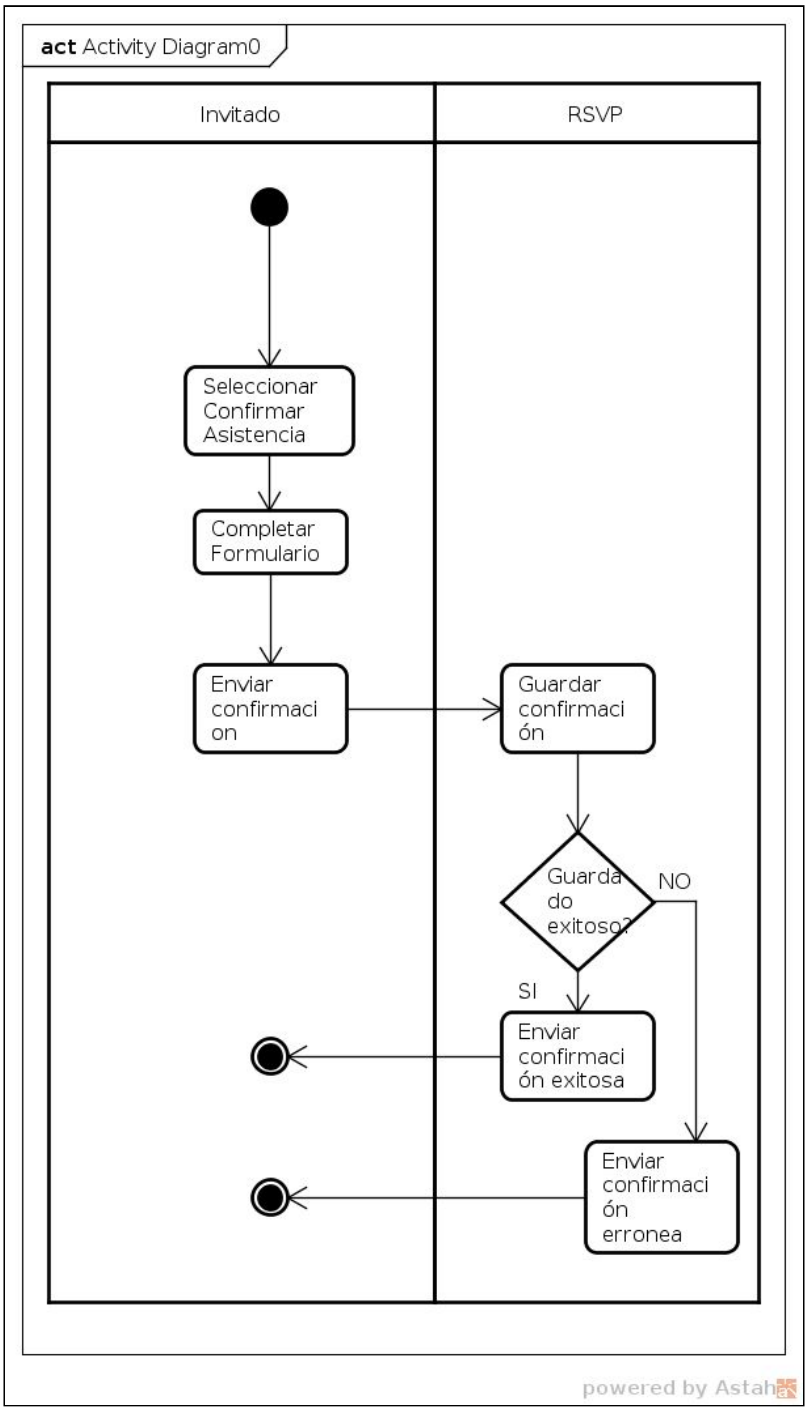
1. El sistema informa por pantalla “No se han encontrado usuarios para la búsqueda solicitada”.
2. El sistema retorna al punto 2 del flujo principal.

## 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.

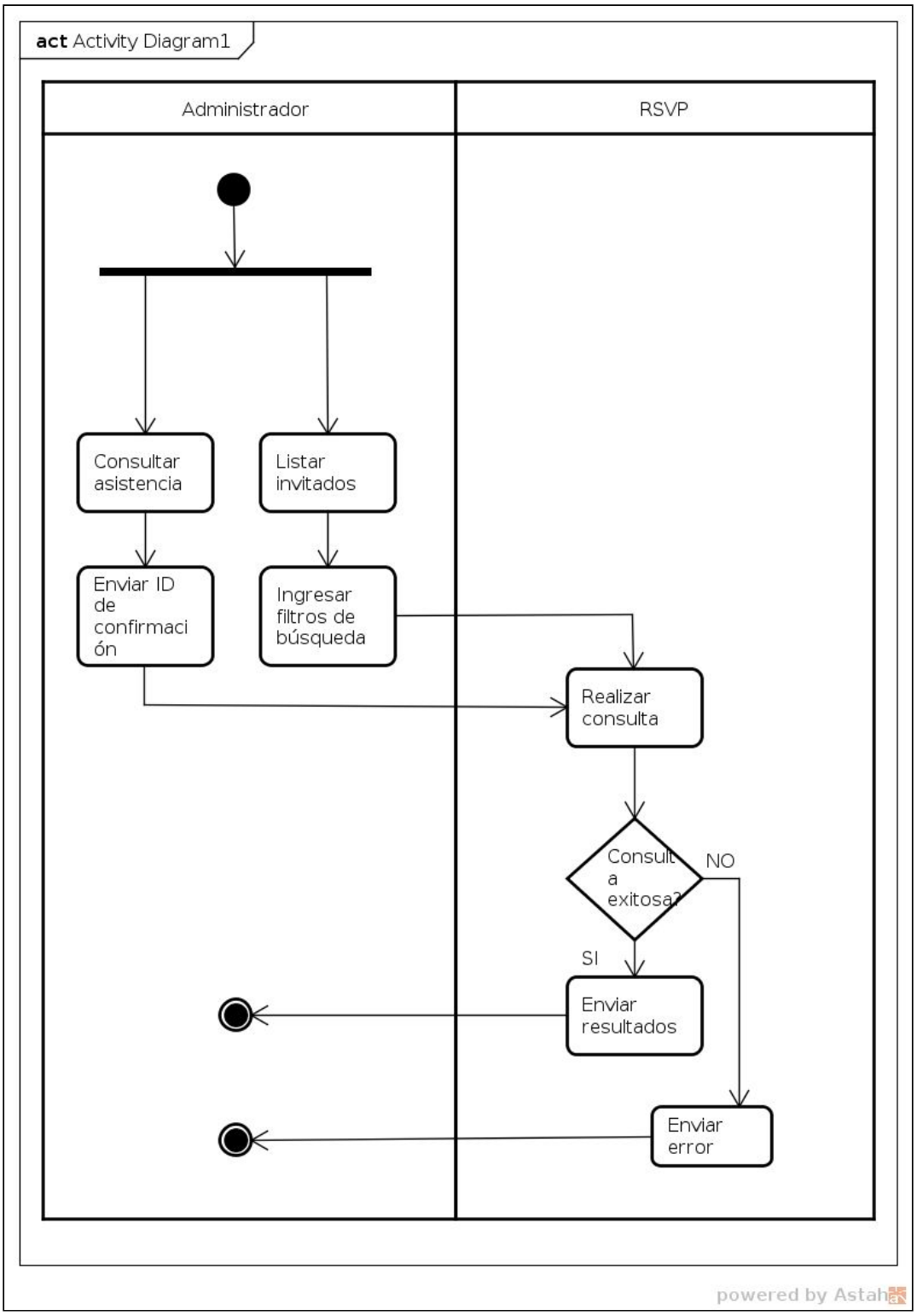
**Escenario**: Confirmación de una asistencia

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




**Escenario**: Consulta de invitados confirmados

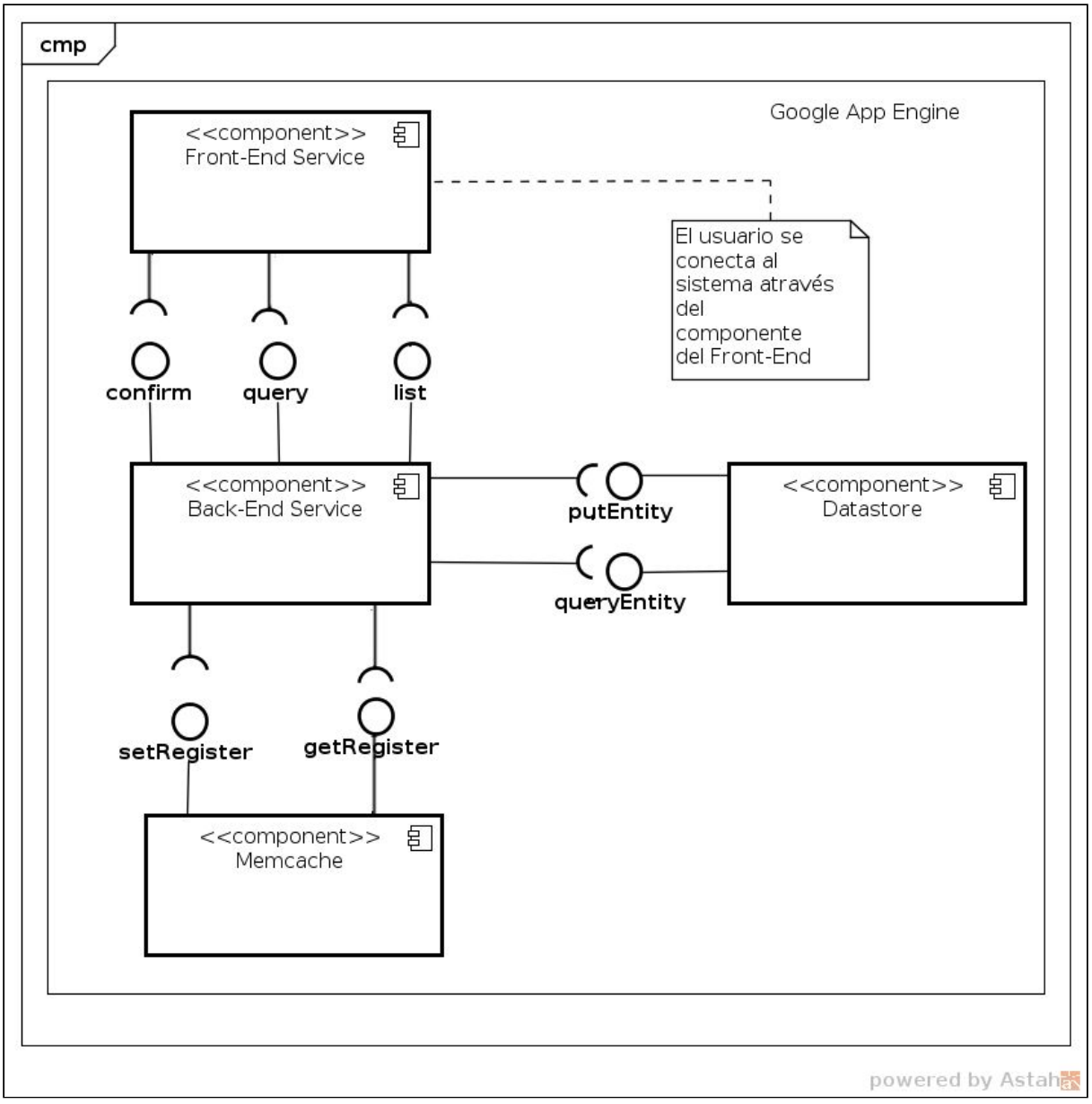
**Descripción**: Flujo de actividades donde se muestran las operaciones que puede realizar una administrador del sistema. La primera es buscar un invitado por su ID de confirmación y la segunda es listar los invitados de acuerdo a algún filtro de búsqueda.




## 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*.




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.

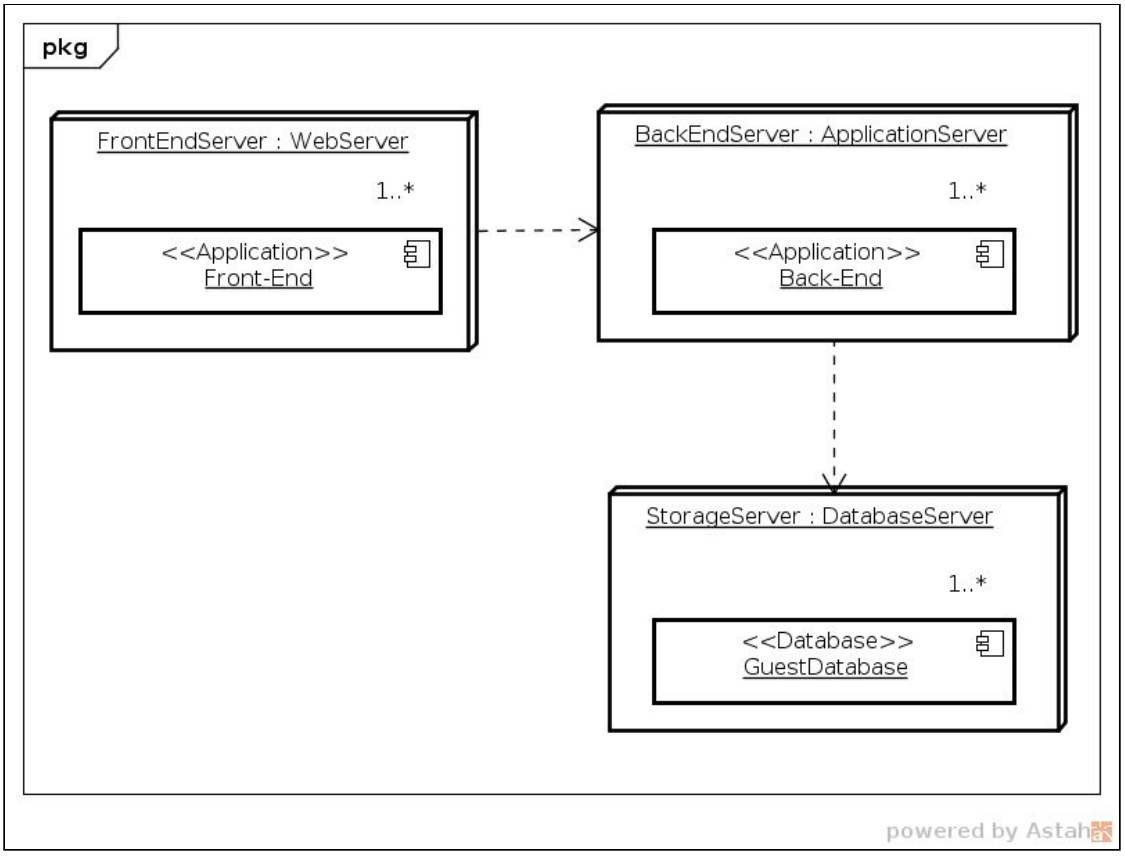
## 2.4. Vista de Física

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.



# 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 - 25/09/2017 | 4 | 26/09/2017 - 27/09/2017 | 8 |
| 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 |

# 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.

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

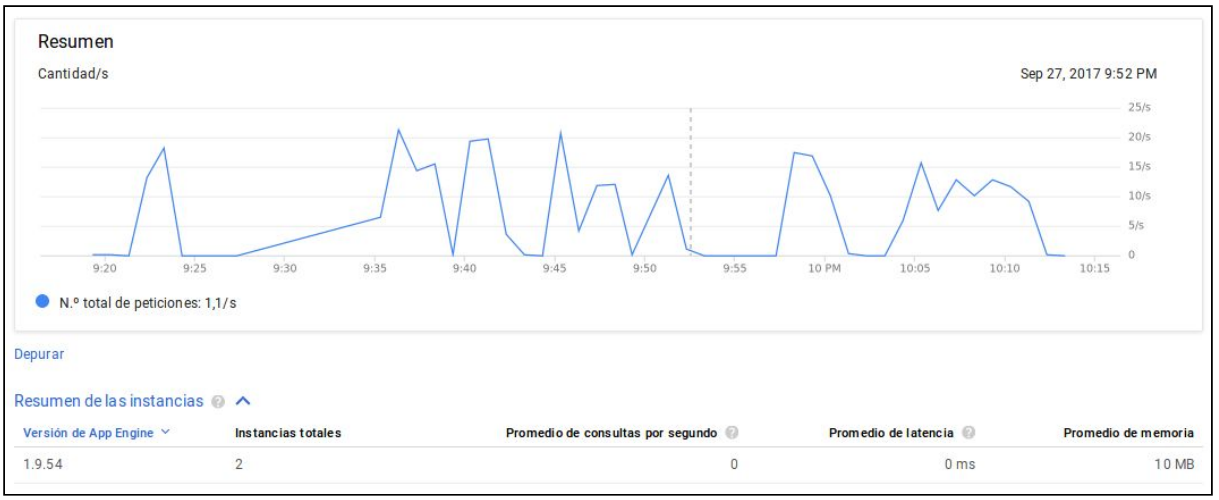
En esta prueba se pretende realizar *requests* sobre el *Front-End* de la aplicación. Como el mismo fue implementado como SPA, la prueba se realizará mediante peticiones GET las cuales descargarán completamente el sitio web a la pc del usuario. El objetivo de este *test* es poder determinar la cantidad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

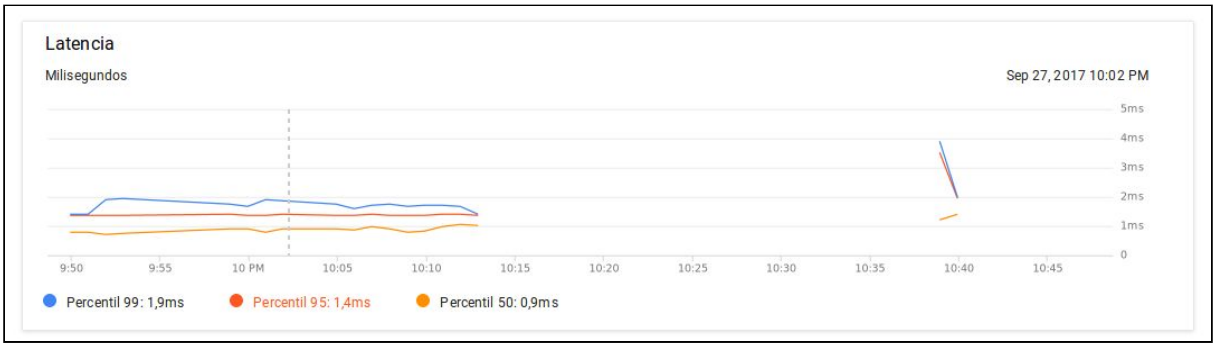
Se pusieron en ejecución grupos de 10, 100, 500, 700, 800, 1000 y 2000 threads constantes durante un tiempo de 30 segundos arrojando los siguientes resultados:

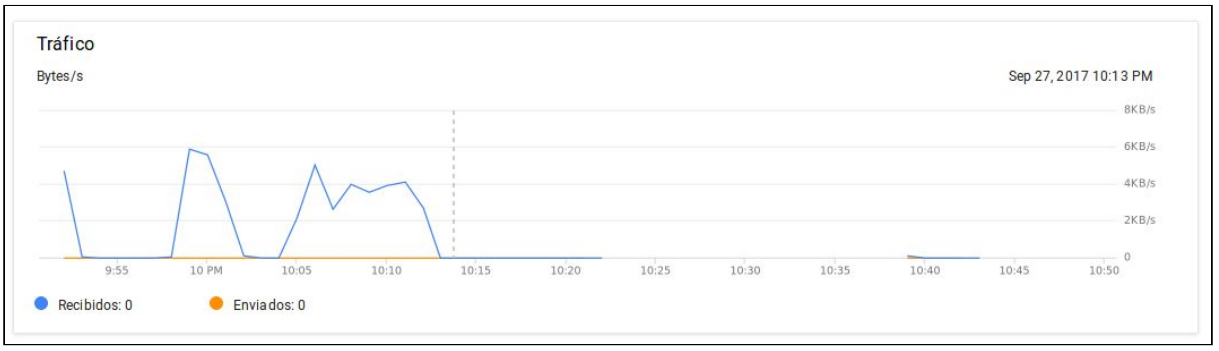
| # Threads | # Muestras | Promedio | Min | Max | Desvío estándar | % Error | Throughput (req/seg) |
|---|---|---|---|---|---|---|---|
| 10 | 1887 | 156 | 76 | 2205 | 198.45 | 0.000% | 62.20742 |
| 100 | 1664 | 1864 | 276 | 31613 | 2113.22 | 0.120% | 38.6061 |
| 500 | 1552 | 13584 | 195 | 62363 | 15972.05 | 12.307% | 24.87857 |
| 700 | 1563 | 17000 | 333 | 70992 | 18403.27 | 10.557% | 22.01501 |
| 800 | 1743 | 19428 | 330 | 65767 | 19688.56 | 29.030% | 26.47287 |
| 1000 | 2201 | 23696 | 326 | 81886 | 26042.08 | 17.583% | 26.7755 |
| 2000 | 10573 | 1658 | 2 | 10799 | 3047.53 | 97.71% | 900.2 |

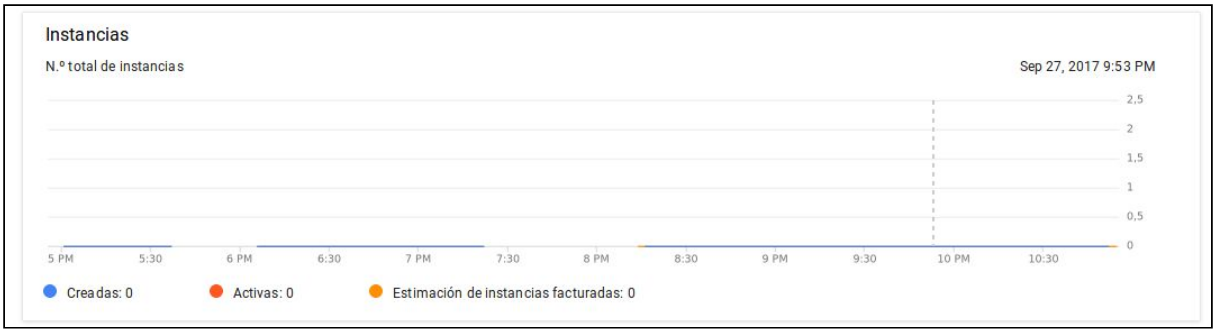
Como se puede observar, el porcentaje de error es aceptable hasta 1000 usuarios concurrentes volviéndose intolerable para 2000. Por otro lado podemos ver que el *throughput* y el tiempo medio de respuesta son muy buenos para 10 usuarios y luego el primero disminuye y el segundo aumenta para cargas de entre 100 y 1000 usuarios pero se mantienen relativamente uniformes alrededor de 25% y 1500 milisegundos respectivamente. Lo mismo sucede con el promedio y el desvío estándar, se puede ver que para pocos usuarios son de milisegundos, luego aumentan al orden de 1 segundo entre 100 y 1000 hilos, para en 2000 pasar a 1.6 y 3 segundos, tiempo que no es malo, pero observando el porcentaje de error, no es un número de usuarios recomendado.

Por el lado de App Engine tenemos los siguientes resultados:




Se puede observar que las peticiones y el tráfico generan picos cuando se ejecutan los *requests* pero la latencia se mantiene uniforme a lo largo del tiempo y las instancias se mantienen nulas, por lo que como el Front-End tiene solamente contenido estático para servir, 1000 usuarios es el máximo de usuarios concurrentes que la plataforma puede atender sin generar una mala experiencia en el usuario.

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

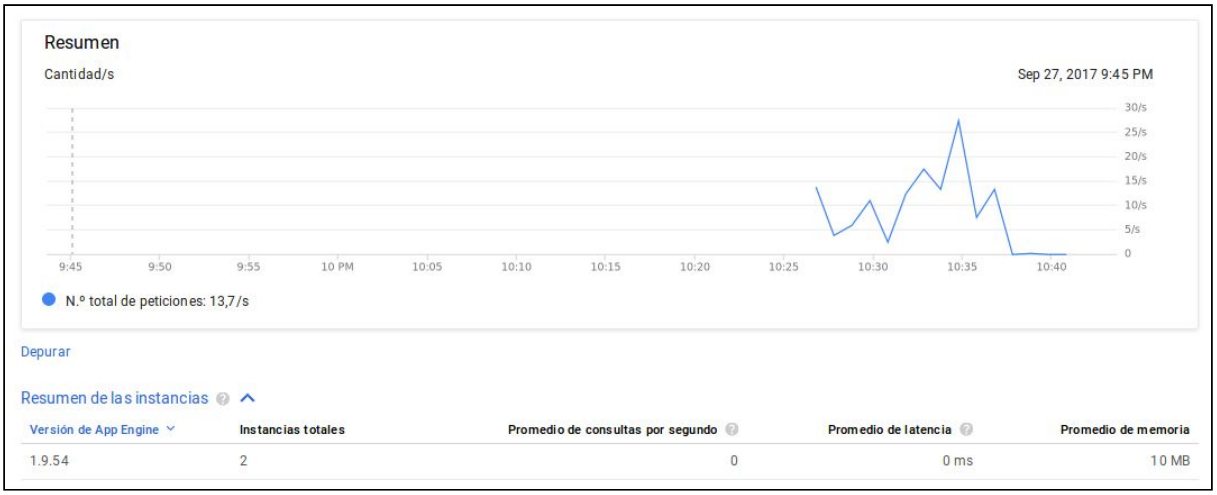
En esta prueba se confirmaciones contra el servicio de *Back-End* de la aplicación. Como este servicio requiere procesamiento y acceso al Datastore por parte de la plataforma de App Engine y teniendo en cuenta el resultado anterior, es de esperar que la cantidad de usuario que pueda soportar operando al mismo tiempo sin generar muchos errores sea menor o igual a 1000. Tener en cuenta que esta operación solo persiste información en la base de datos, no consulta.

Se pusieron en ejecución grupos de 10, 100, 500, 700 y 1000 threads constantes durante un tiempo de 30 segundos confirmando asistencias al evento, arrojando los siguientes resultados:

| # Threads | # Muestras | Promedio | Min | Max | Desvío estándar | % Error | Throughput (req/seg) |
|---|---|---|---|---|---|---|---|
| 10 | 823 | 116 | 95 | 716 | 45.02 | 0.000% | 81.62253 |
| 100 | 470 | 2353 | 357 | 10732 | 1809.75 | 0.213% | 30.45619 |
| 500 | 799 | 10435 | 368 | 33089 | 8441.2 | 4.005% | 23.06049 |
| 700 | 1727 | 16167 | 356 | 46312 | 14804.17 | 29.068% | 37.202119 |
| 1000 | 11749 | 4111 | 1 | 59540 | 12712.86 | 86.875% | 197.2665 |

Como puede verse en la tabla anterior, en este caso y teniendo en cuenta que se está enviando información sensible, el porcentaje de error es bueno hasta 500 usuario concurrentes realizando la confirmación, es aceptable con 700 usuarios y es intolerable con 1000 o más. Por otro lado podemos observar que el *throughput* es muy bueno para 10 usuarios, se mantiene uniforme alrededor de 30 *requests* por segundo para usuarios entre 100 y 700, pero tanto el promedio como el desvío estándar hasta 5 u 8 veces cuando se pasa de 100 a 500 y 700 usuarios por lo que; teniendo en cuenta el tiempo promedio que debe esperar un usuario para que su confirmación sea recibida, un total de 100 usuarios concurrentes sería un valor aceptable para esta prueba.

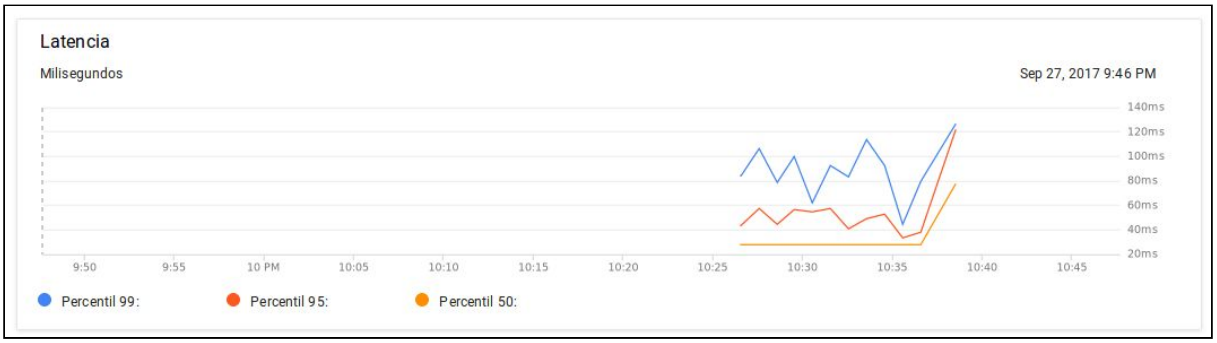
Por el lado de App Engine se obtuvieron los siguientes resultados:

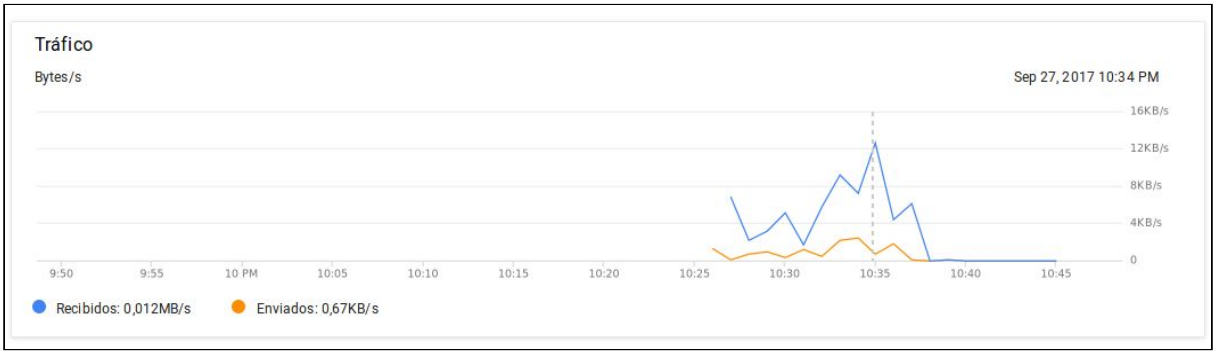


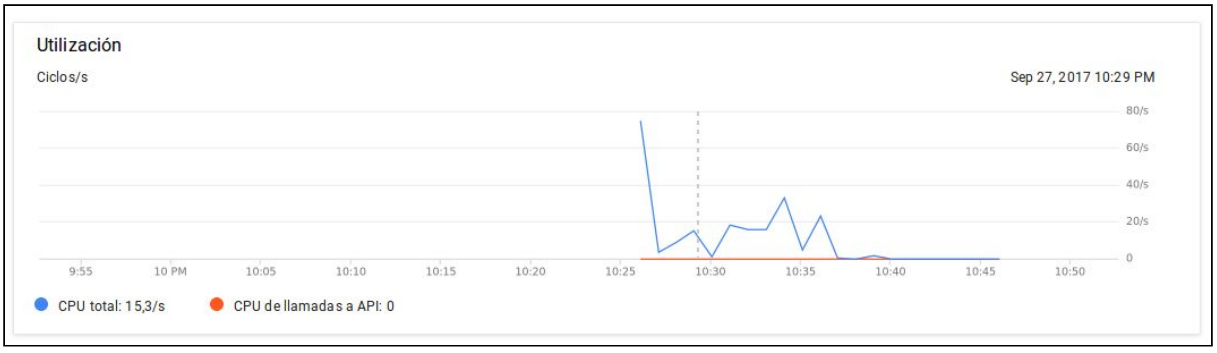

Depurar

Resumen de las instancias

| Versión de App Engine | Instancias totales | Promedio de consultas por segundo | Promedio de latencia | Promedio de memoria |
|---|---|---|---|---|
| 1.9.54 | 2 | 0 | 0 ms | 10 MB |




En los gráficos expuestos, en el primero de ellos se puede ver que el pico máximo de peticiones es de alrededor de 30 por segundo que puede asociarse con los valores obtenidos en JMeter para la corrida de 100 threads el cual dió un throughput similar con una tasa relativamente baja de errores. También al mismo valor se puede asociar el pico de latencia de 120 ms obtenido en el segundo gráfico presentado. Por otro lado en el tercero de ellos se puede observar que en el mismo tiempo se produjo la mayor tasa de transferencia de datos de alrededor de 12 KB/s. Por último; en los gráficos 4 y 5, se ve que desde 10:25 hasta 10:40 la utilización de las instancias levantadas por App Engine tuvieron movimiento debido a las pruebas realizadas. En el gráfico de Instancias se puede ver que alrededor de 10:35 se activó una de las instancias para responder las peticiones que se hicieron durante las pruebas y que coincide con el aumento en los ciclos de utilización de la misma.

## 4.3. Prueba N° 3: Test de carga sobre el Datastore y la Memcache

El objetivo de esta prueba es mostrar que la utilización de la Memcache como storage intermedio mejora los tiempos de respuesta de las peticiones. Según la documentación de App Engine, el tamaño de esta *cache* es de alrededor de un 1MB; por lo cual, se cuenta con el espacio suficiente para realizar el *test* ya que las entidades pesan en promedio 234 Bytes.

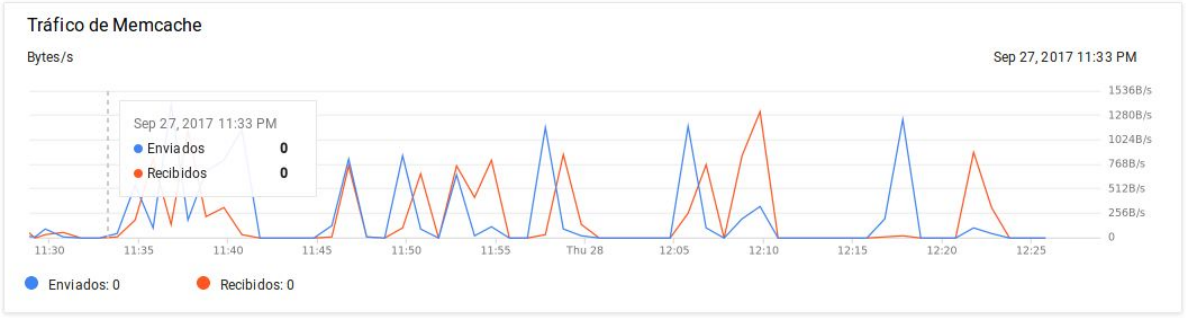
Se ejecutaron 3 pruebas con grupos de *threads* de 30, 400 y 500 usuarios realizando consultas al endpoint /query?id=”{ID}” de la siguiente manera: con la *cache* vacía por cada prueba se realiza una consulta inicial para cargar la cache y se miden los tiempos, luego se realiza una segunda consulta con los mismos parámetros y se miden los tiempos nuevamente para realizar la comparación. Se asume que el tiempo de carga de la *cache* es despreciable frente al tiempo de consulta del datastore por lo que el tiempo consumido por el request de carga se debe al acceso a este último.

| # Threads | Carga inicial | Promedio | Min | Max | Desvío estándar | % Error | Throughput (req/seg) |
|---|---|---|---|---|---|---|---|

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 30 | Si | 1067 | 297 | 2085 | 509.73 | 0.000% | 2.83473 |
| 30 | No | 302 | 282 | 414 | 28.66 | 0.000% | 3.00541 |
| 400 | Si | 2587 | 454 | 9866 | 1494.04 | 0.000% | 26.04336 |
| 400 | No | 2270 | 421 | 9577 | 1343.55 | 0.000% | 26.36088 |
| 500 | Si | 4406 | 859 | 17091 | 2783.13 | 1.200% | 22.33539 |
| 500 | No | 3905 | 355 | 15783 | 2520.27 | 0.400% | 24.20136 |

Como puede observarse en todos los casos; pero más notoriamente en el primero de ellos, se produce una mejora sustancial en los tiempos de respuesta para los *requests* que son respondidos utilizando la *cache* de datos de App Engine de hasta un orden de magnitud, no viéndose el mismo resultado para una mayor cantidad de carga en el sistema donde se puede apreciar que la diferencia entre usar la cache y no utilizarla es de algunos milisegundos y aunque esto no parezca demasiado se puede observar que en el caso para 500 usuarios tanto el porcentaje de error y el *througputh* bajaron seguramente debido a que como ya no se realizan tantas consultas al datastore, no cancelan tantos requerimientos por verse este último y la instancia colapsados de consultas ya que al utilizar la *memcache* las mismas se resuelven más rápido y de la misma manera descarga la aplicación.

Por el lado de App Engine se pueden observar las siguientes estadísticas para la corrida de 30 usuarios:


En la primera de ellas se puede ver que que el porcentaje de visitas a la *cache* es del 97.24% de visitas por lo que casi todos los *rquests* fueron servidos por datos existentes en esta última.

Por otro lado; en el segundo gráfico, de entre todos los picos que se muestran se puede apreciar que entre las 12:15 y las 12:20 se cargó la *memcache* con datos pertenecientes a la primer corrida con la misma vacía y entre 12:20 y 12:25 se volvió a correr la misma prueba que demuestra que casi todos los datos fueron servidos por esta.

Por lo cual, al realizar aplicaciones que necesiten guardar un volumen grande de datos para luego ser consultados es necesario hacer uso de esta herramienta ya que; como se demostró anteriormente, mejora la performance del sistema.

# 5. Código fuente

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