Trabajo Práctico 1

Documento de Arquitectura

75.61 Taller de Programacion III

Segundo Cuatrimestre 2021 - FIUBA

Nombre y Apellido	Padrón
Franco Giordano	100608

Objetivos

Se solicita el desarrollo de un sitio web institucional con los siguientes requerimientos funcionales:

- Se debe registrar las visitas en las distintas páginas: home, jobs, about, about/legals y about/offices.
- El conteo de visitas debe ser incremental y por cada página de forma individual.
- En cada sección se debe visualizar el contador de sus visitas totales recibidas.

Como requerimientos no funcionales, tenemos los siguientes:

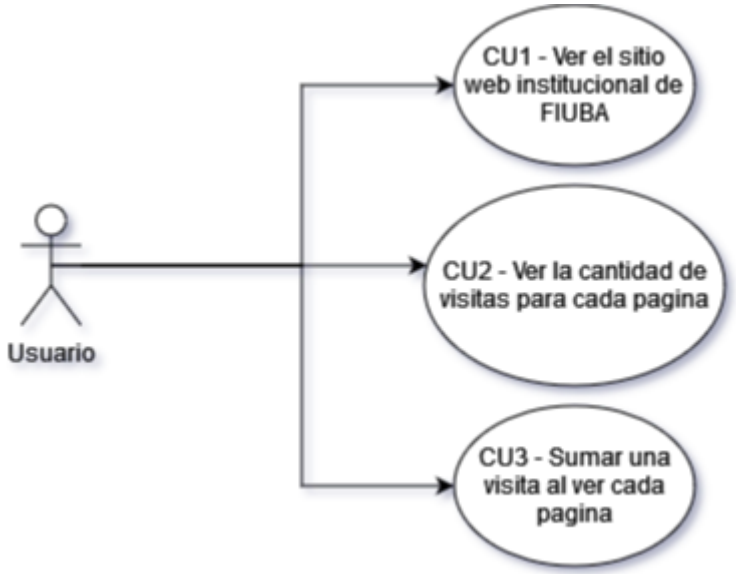
- El sistema debe escalar automáticamente con el tráfico recibido.
- El sistema debe mostrar alta disponibilidad hacia los clientes.
- El sistema debe ser tolerante a fallos como la caída de procesos.

El presente documento analiza en detalle la última arquitectura planteada por el Reporte de Pruebas de Carga (Iteración 3).

El objetivo ahora será entender la arquitectura planteada y sus decisiones de diseño.

Casos de Uso

Antes de comenzar, entenderemos el propósito del sistema mediante algunos casos de uso.



Detallamos los mismos a continuación:

CU1 - Ver el sitio web institucional de FIUBA
Descripción: El usuario visualiza una de las 5 páginas del sitio web.
Actores Participantes: Usuario
Pre Condiciones: La página que desea ver debe existir.
Post Condiciones: Obtiene la página web pedida (por ej: home)

CU2 - Ver la cantidad de visitas para cada página
Descripción: El usuario obtiene la cantidad total de visitas hechas a la página actual, globalmente.
Actores Participantes: Usuario
Pre Condiciones: La página a consultar debe existir.
Post Condiciones: Obtiene el contador de visitas de la página web pedida (por ej: home)

CU3 - Sumar una visita al ver cada página
Descripción: El usuario registra su visita en una de las 5 páginas del sitio web.
Actores Participantes: Usuario
Pre Condiciones: La página que desea registrar debe existir.
Post Condiciones: Se registra su visita para esa página web.

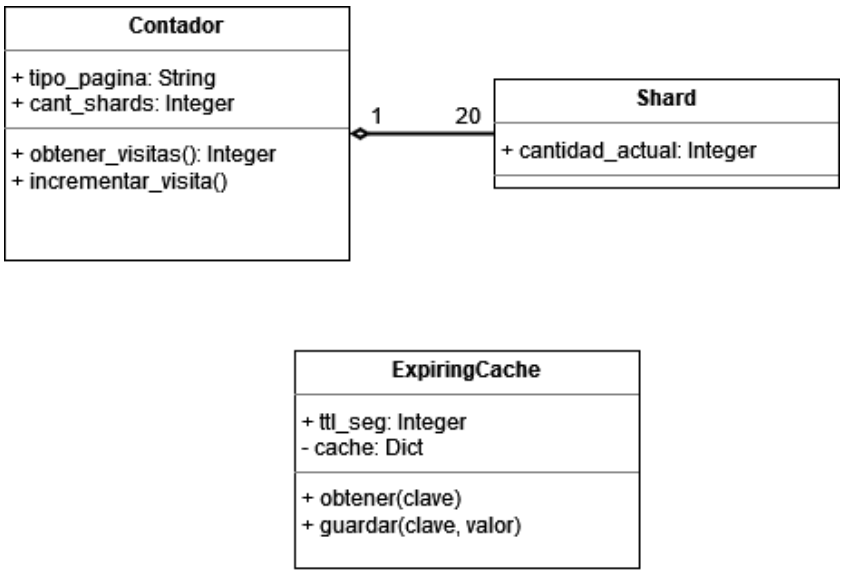
Buscamos un sitio web con 5 páginas: home, jobs, about, about/legals y about/offices. El mismo debe ofrecer un contador de visitas para cada página, mostrando el acumulado hasta el momento y registrando visitas nuevas.

Vista Lógica

Entendiendo que usos intenta resolver nuestro sistema, vemos ahora la relación de los conceptos a un alto nivel.

Diagrama de Clases

Siendo que el sistema es casi en su totalidad funcional, tenemos muy pocas clases en nuestro dominio:



Tanto Contador como Shard son utilizadas para representar los contadores en memoria. En particular, Contador abstrae la lógica de agregación de los Shards para una página dada. Esto quiere decir, que tanto en memoria como en la base de datos, tenemos un contador (y colección) distinta para cada página, almacenando en la misma 20 shards independientes.

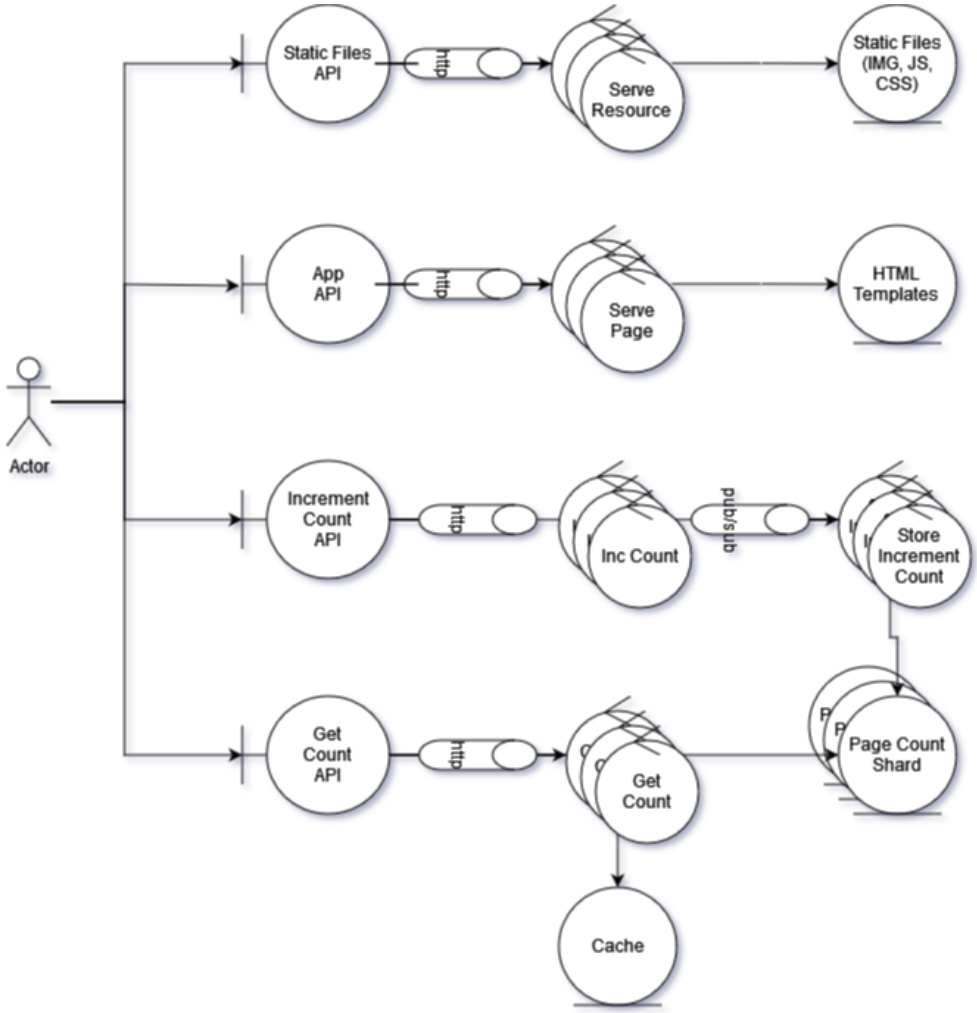
Por otro lado, ExpiringCache ayuda a mantener en memoria el caché de los Contadores ya computados, donde cada clave tendrá un time-to-live independiente.

Vista Física

Buscaremos ahora comprender la topología del sistema planteado, desde las principales unidades de cómputo pensadas hasta el despliegue concreto efectuado.

Diagrama de Robustez

Como ya adelantamos más arriba, estudiaremos la última arquitectura planteada por el Reporte. La misma es un sistema distribuido pensado para soportar altas cargas:



Como vemos, el usuario interactúa con el mismo mediante cuatro interfaces HTTP:

- App API: Punto de entrada para el usuario. Dado un query param, devuelve el HTML correspondiente (por ej: home.html)
- Static Files API: El HTML anterior busca los recursos mediante esta API, como las imágenes, CSS y Javascript de la web.
- Get Count API: Dado un nombre de página mediante query param, devuelve la cantidad de visitas acumuladas para la misma.
- Increment Count API: Dado un nombre de página mediante query param, registra la visita en el acumulado correspondiente.

Notar que todos los controladores corren en paralelo para mejorar la performance. En cierta medida, lo mismo ocurre con la base de datos la cual está particionada para cada página con cierto factor de sharding. Con esto último buscamos mejorar la tasa de escritura.

Diagrama de Despliegue

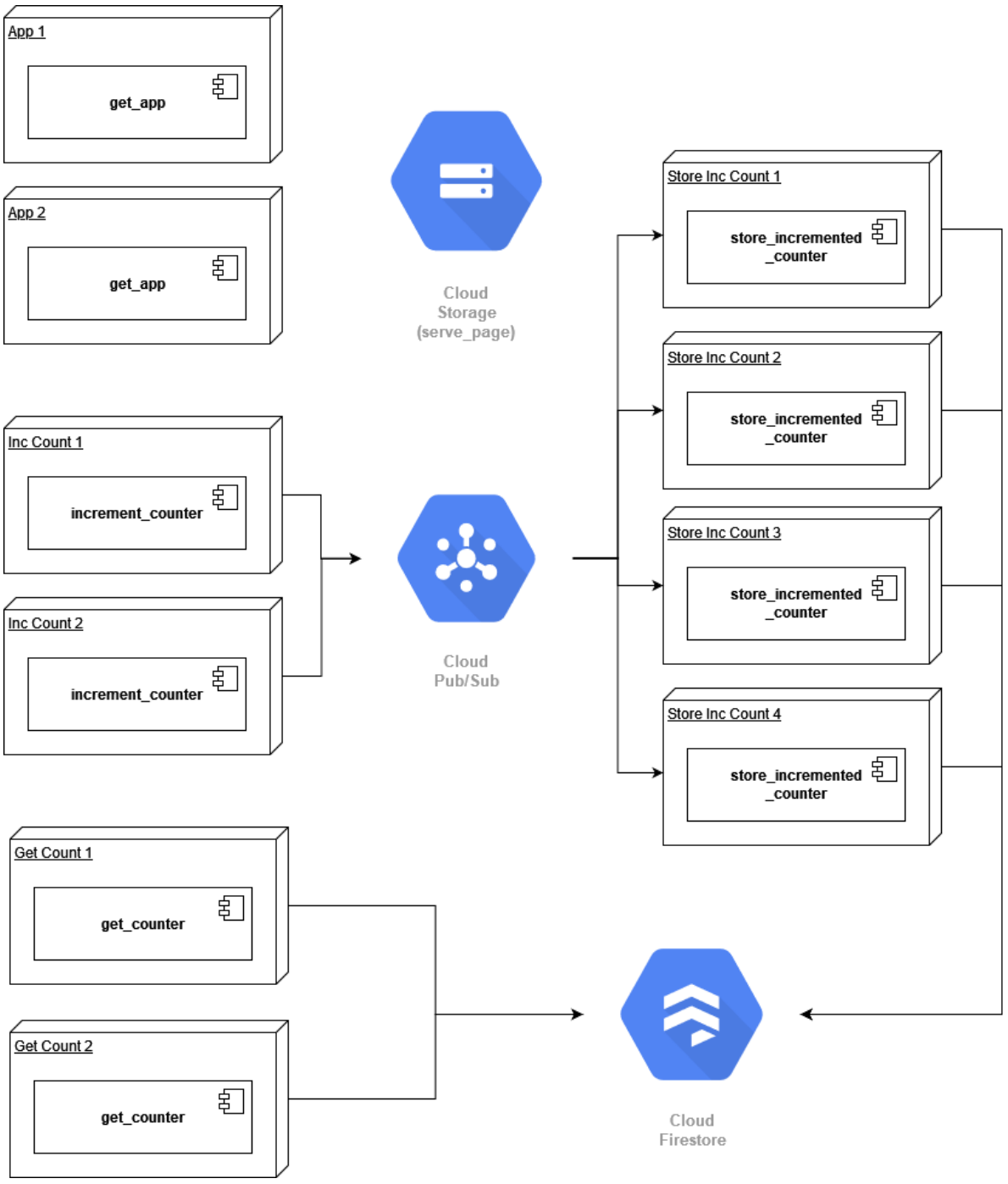
Veremos ahora como distribuimos físicamente los nodos en las máquinas de GCP. En primer lugar, cabe destacar los servicios utilizados para cada funcionalidad:

- Cloud Storage: almacenamiento y servicio de archivos estáticos (img, css, js).
- Cloud Functions: Ejecucion serverless de `app`, `get-counter`, `inc-counter` y `store-inc-counter`.
- Cloud Pub/Sub: publicación de mensajes en tópicos para la comunicación asincrónica de `inc-counter` y `store-inc-counter`.
- Firestore Nativo: Almacenamiento de los contadores de visitas para cada página.

Siendo que estos servicios abstraen muchos detalles de su implementación, los representaremos en el diagrama no como computadoras o procesos sino como servicios externos. Denotaremos aun así las instancias de las Cloud Functions, con el detalle de que representan instancias *virtuales* pues Google Cloud no nos asegura recursos privados para estos servicios.

Para implementar el caché de get-counter se utilizó simplemente una variable global a la instancia, *no* se hizo uso de servicios como Memorystore.

Desplegamos entonces nuestro sistema y lo diagramamos en la siguiente página.



Los nodos virtuales funcionan mediante Cloud Functions con Python 3.9. Notar como los nodos App y el servicio Cloud Storage no se comunican con otros nodos del sistema, pues lo hacen directamente con el cliente.

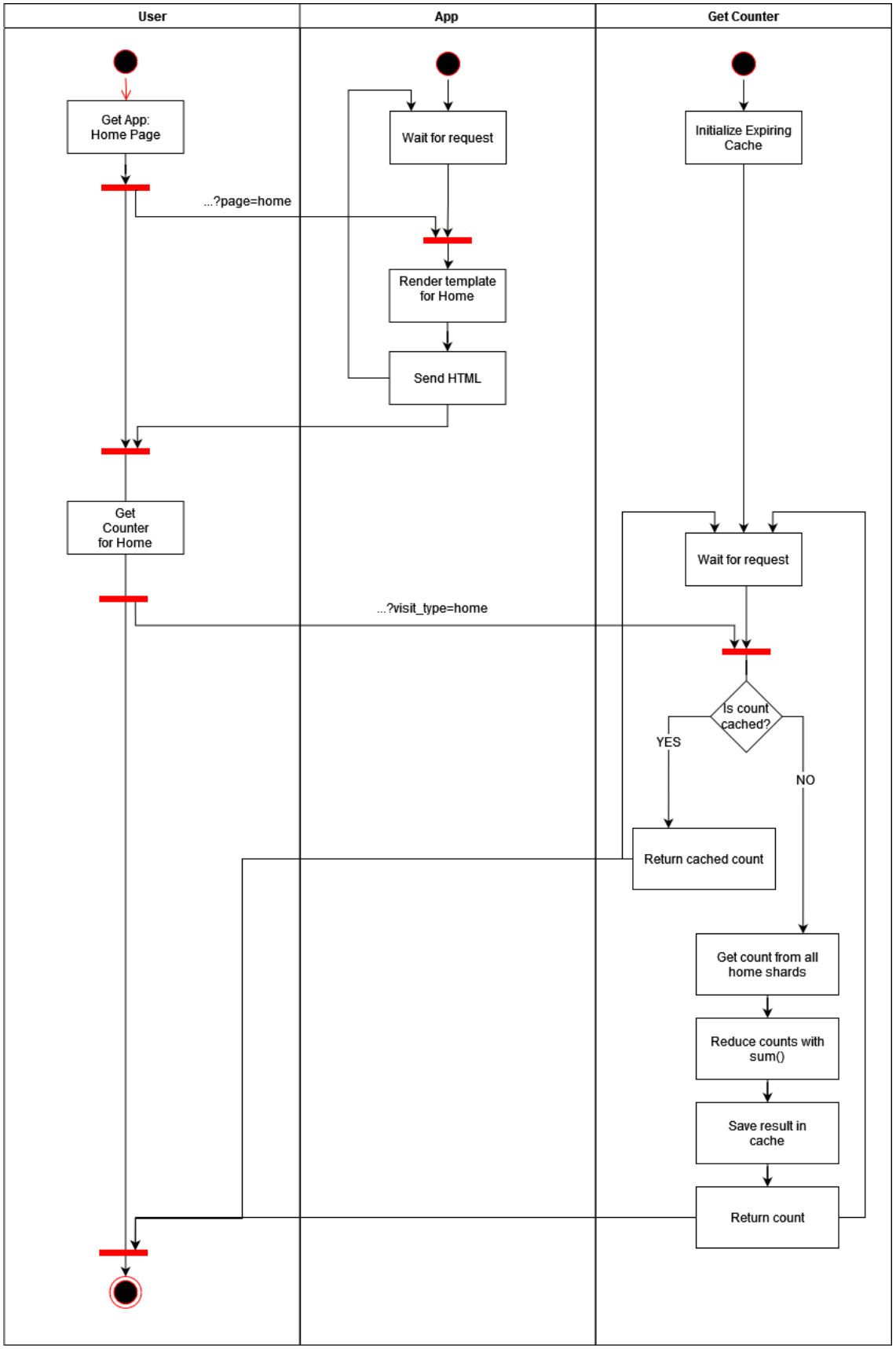
Vemos cómo impactan las decisiones de instancias: tenemos 2 instancias para cada Cloud Function, salvo Store Inc Counter que posee 4.

Vista de Procesos

Estudiaremos ahora la comunicación del sistema al momento de ejecutarlo, observando los flujos dentro del mismo. Notar que muchos de estos procesos se muestran de forma simplificada para facilitar la lectura.

Encontraremos el primer diagrama de actividad en la siguiente página.

Diagrama de Actividad - Obtener Contador

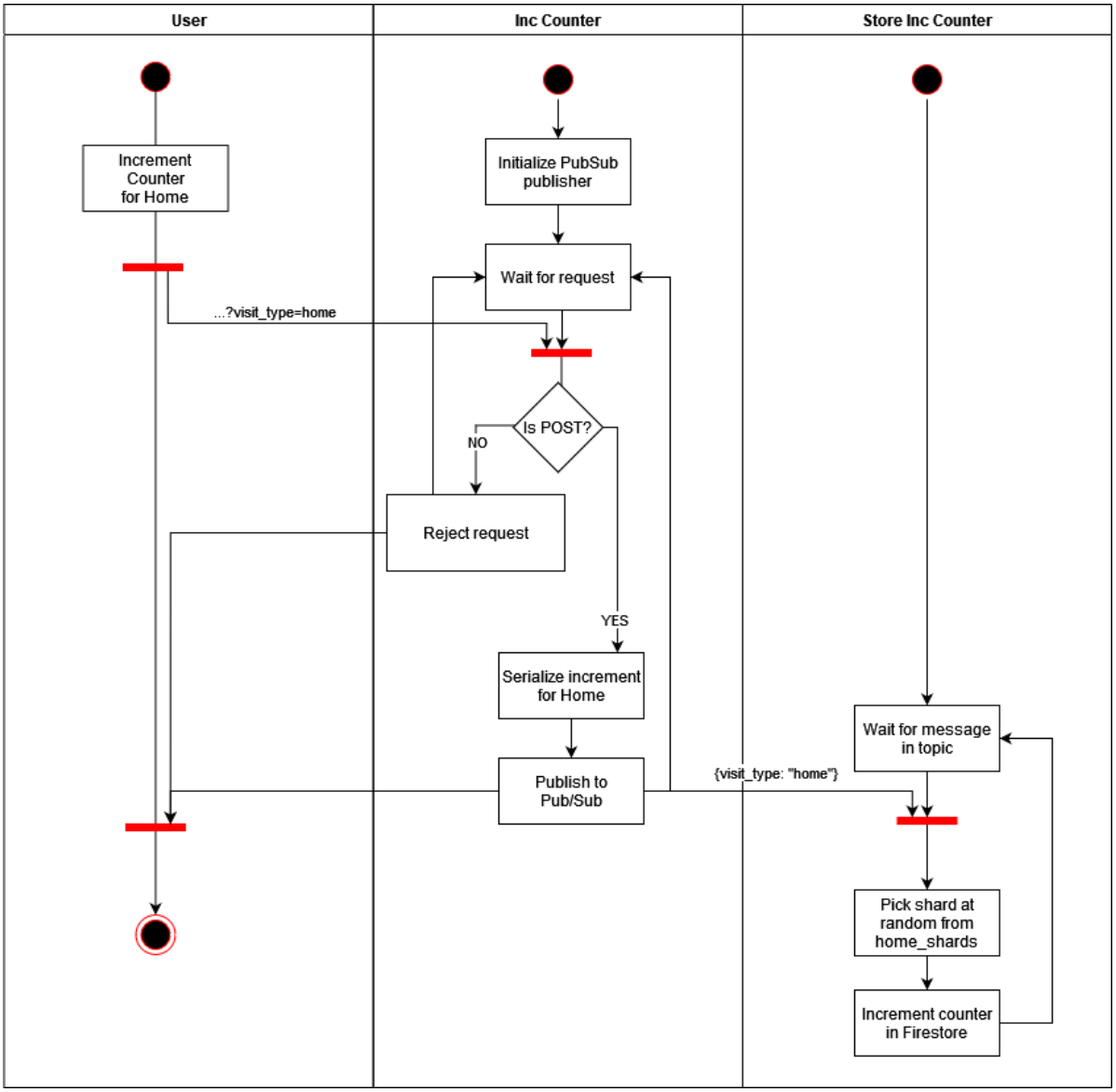


Vemos en una primera instancia *sólo* el proceso de buscar la web HTML y obtener el contador correspondiente. Notar el pase de parámetros mediante query params para todos los endpoints HTTP, y el uso de un caché con timeout en Get Counter.

El timeout del cache es configurable mediante variable de entorno, escogiendo para este despliegue un timeout de 10 segundos para cada página.

Diagrama de Actividad - Incremento de Contador

Vemos ahora un proceso que antes ocultamos: el incremento de un contador de la web.

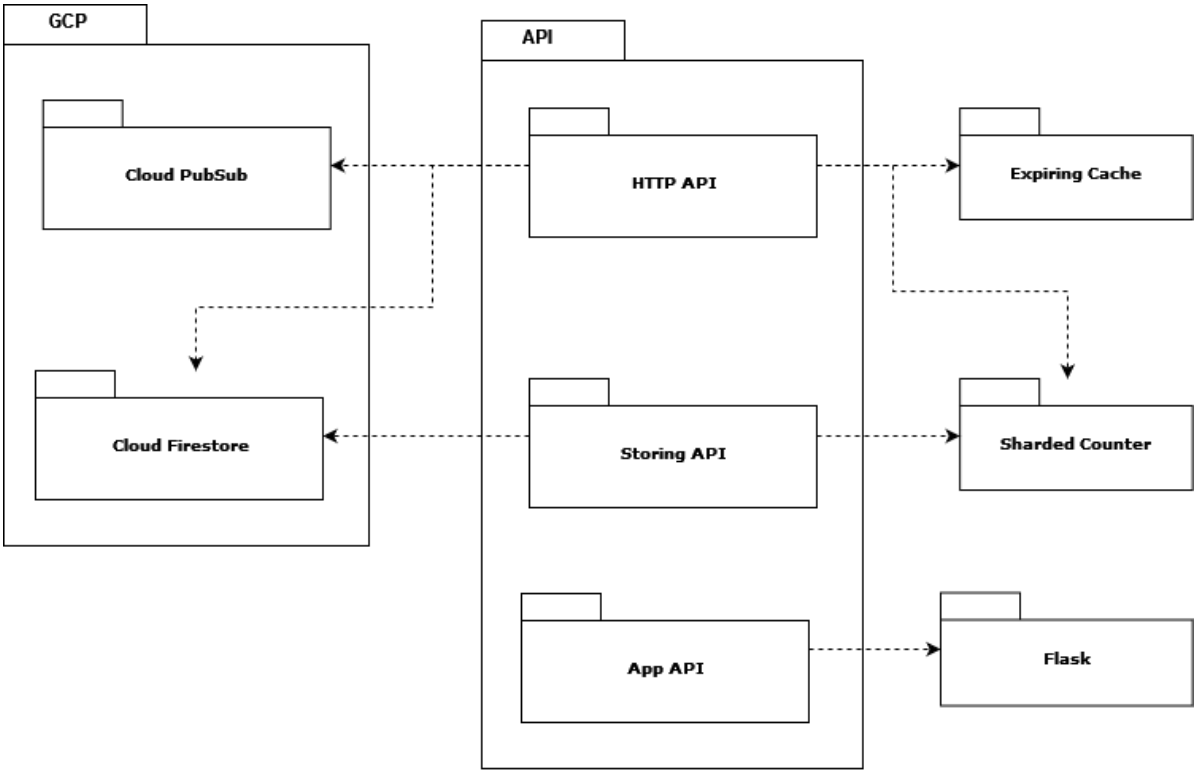


Nuevamente, vemos el uso de query params para el endpoint HTTP. La selección de un número de shard es aleatorio, pudiendo elegir entre 20 shards distintos para una página dada.

Vista de Desarrollo

Veremos ahora, a grandes rasgos, los paquetes utilizados por la implementación y la organización de los mismos.

Diagrama de Paquetes



Observamos que nuestra API está dividida en 3 grandes grupos App (sirve el HTML), HTTP (obtiene el contador y lo incrementa), y Storing (suscripta a PubSub, incrementa el shard en la BDD). Además, tanto Storing como HTTP requieren paquetes de Google Cloud, mientras que App utiliza Flask para cargar los HTML de disco.

Trabajo a Futuro

Si bien el sistema presentado resuelve los casos de uso propuestos, quedan algunas mejoras para hacer a futuro:

- Realizar más validaciones de parámetros: si se pide por una página inexistente el sistema arroja un error no muy amigable al usuario. Además, la Storing API intenta incrementar el shard inexistente y dará error, lo cual no interrumpe el servicio pero es evitable.
- Evaluar opciones de Batching al incrementar contadores: Actualmente todos los incrementos se realizan en forma individual. Como se sugirió en el Reporte, sería valioso estudiar un esquema de Batching para acelerar los tiempos, intentando siempre de adaptarse al entorno stateless de las Cloud Function.
- Implementar una lógica de retries en el cliente: Actualmente, si falla un incremento el cliente no reintenta el registro. Bastaría con agregar, por ejemplo, 3 reintentos y analizar su efecto con altas cargas.