Pontificia Universidad Católica de Chile
Departamento de Ciencia de la Computación
IIC2173 Arquitectura de Sistemas de Software

Documentación Entrega 1

Grupo 4

---

1. Trabajo Delegado

Para el trabajo delegado se siguió el siguiente tutorial

En el cual mediante el uso de las *lambda functions* se crea un procedimiento que analiza el sentimiento promedio del texto. para el análisis se ocupa *amazon comprehend*, un servicio de machine learning para manejar datos no estructurados. Por ultimo para llamar al procedimiento se ocupa a *Api Gateway*, que nos permite mediante *http request* llamar a nuestra función.

Por lo tanto el flujo con los tres casos de uso queda asi:

- En, primer lugar se hace un request a la Api Gateway, en el siguiente endpoint: https://qzd76d6xkk.execute-api.us-east-2.amazonaws.com/default/sentiment. este debe ser del tipo POST y en el body recibe el atributo text. que sera el texto a analizar. Aquí se decidió activarla con http ya que era lo mas directo para realizar desde nuestro backend, y se dejo la función abierta sin seguridad.
- El Api Gateway llama a la función lambda, la cual es código que puede correr sin la necesidad de manejar un servidor. en esta función se llama al servicio Comprehend y se retorna su respuesta.
- Ocupamos el servicio Comprehend para dos casos, en primer lugar extraemos el idioma del texto, y luego sabiendo el idioma podemos extraer el sentimiento general de el texto. este es devuelto para ser retornado por la lambda como respuesta.

Cabe destacar que este flujo después vuelve a modo de callback con la respuesta llegando así al response luego de realizar un request.

2. Mensajes en tiempo real

RF1

Para lograr que los mensajes se muestren sin tener que refrescar la página utilizamos web Sockets y serializamos los mensajes a JSON. El frontend obtiene la información que ingresa

un usuario, la envía al backend, este la procesa, la guarda y la envía de vuelta al frontend para ser mostrada en la aplicación a todos los usuarios de la sala del chat.

RF2

Para este requisito se hizo lo siguiente:

- En el backend, luego del proceso de censurar mensajes, si es que encuentra un match con @ se intenta encontrar el usuario seguido del @. Si se encuentra, se obtiene su email.
- Se utilizó el siguiente código para implementar nodemailer. Con esto, se envía un correo al mail del usuario encontrado en la parte anterior, con el mensaje en el cual fue etiquetado. Para esto se utilizó una cuenta de gmail creada solo para este fin.

Hay que tener las siguiente consideraciones:

- Los mensajes censurados no se notifican. Por ejemplo: "Hola @Valentina me siento triste" se mostrará en le chat como censurado, pero no se enviará un correo a Valentina.
- Si se etiqueta a un usuario no válido, el mensaje de mostrará, pero no se notificará. Por ejemplo, "Hola @Juan", cuando Juan no es un nombre de usuario registrado, mostrará el mensaje en el chat, pero no enviará ningún correo.

```
function (){
    return }
```

3. Caché

RF1

Para este requisito se utilizo el servicio ElastiCache for Redis de AWS, que permite acceso solo a las instancias que se encuentran dentro de el security group dispuesto para el Auto Scaling Group.

RF2

Se almacenó la información en caché para los siguientes casos de uso:

- Mensajes por grupo: Cada vez que un usuario entra a leer un grupo este solicita todos los mensajes del mismo, si la información no esta en caché se solicita a la base de datos (MongoDB), y se guardan, utilizando el id del grupo como llave, todos los mensajes en una lista, fijando el tiempo de expiración en 10 minutos. Cada vez que alguien manda un mensaje, se revisa si la información del grupo al que el mensaje esta en caché, si

es que esta se agrega el mensaje al final de la lista y se vuelve a setear el tiempo de expiración a diez minutos, de esta forma si es que un grupo tiene sus mensajes en caché pero no se envía ningún mensaje a través de este durante 10 minutos, la información se elimina del caché y el próximo request va a ir a parar a la Base de Datos. Se decidió utilizar la lista como estructura de datos para almacenar la información de los mensajes para poder mantener estos en orden por fecha de envío, simplemente agregando a al final los nuevos mensajes en O(1).