Fuego de Quasar – MeLi challenge.

Información acerca de los repositorios y el conector a la Base de datos utilizado:

- **mongo connector:** mongodb+srv://ximbayer:eav-1234@clustermeli.89pnn.mongodb.net/myFirstDatabase?retryWrites=true&w=majority
- **nombre BD en mongo:** fuegodequasar

- **Git Repository:** https://github.com/ximbayer/Fuego-de-Quasar.git

- **Heroku Repository**:

  name: fuego-quasar-lcarreras (**repo**: https://git.heroku.com/fuego-quasar-lcarreras.git)

  URL API in Heroku: https://fuego-quasar-lcarreras.herokuapp.com/

- **Endpoints**:

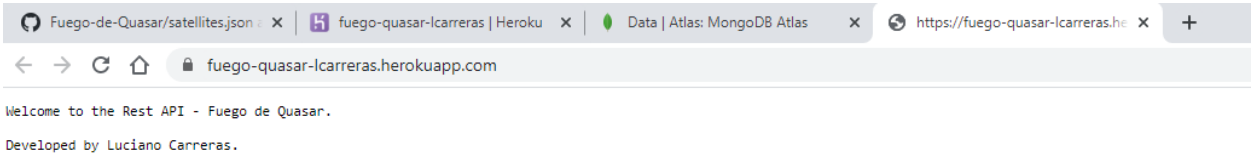
  https://fuego-quasar-lcarreras.herokuapp.com/

  https://fuego-quasar-lcarreras.herokuapp.com/topsecret

  https://fuego-quasar-lcarreras.herokuapp.com/topsecret_split

  https://fuego-quasar-lcarreras.herokuapp.com/topsecret_split/{satellite_name}

- **Atencion! -> router := mux.NewRouter().StrictSlash(true)**



Aclaraciones y comentarios a tener en cuenta:

GetLocation:

Para la función "GetLocation" creada, se utilizó la teoría encontrada en Wikipedia: https://es.wikipedia.org/wiki/Trilateraci%C3%B3n

Además, se utilizaron funciones matemáticas y geométricas obtenidas desde la documentación de GO encontrada en la red. No se importaron líneas de código directamente, sino que se adaptaron a las necesidades que nos mostraban las fórmulas de Wikipedia como por ejemplo calcular d, j, I, que son las distancias a los centroides de los círculos creados a partir de la coordenada y la distancia a la nave, donde dicha distancia para nosotros es el radio de cada circulo.

GetMessage:

Para la función "GetMessage" creada, se tuvo en cuenta un desfasaje "hacia la derecha". Es decir, a partir de encontrar el **slice** de strings más corto, se analiza con el largo de los **slice** restantes. Si el largo de un mensaje es mayor, se considera a analizar las palabras a partir de la ubicación "i + desfasaje".

Es decir, si el 1er mensaje tiene un largo de 4 palabras, y el largo mínimo encontrado (entre todos los mensajes) es de 3, al 1er mensaje se lo comienza a analizar en su contenido a partir de la "posición 1", y no desde la "posición 0" (del slice), dado el desfasaje calculado por el desarrollador.

A tener en cuenta para el servicio "topsecret_split", del punto 3:

Para el servicio topsecret_split se tuvo en cuenta que el listado de llamadas a los satelites se inicia vacío.

Es decir que NO es la misma que para el servicio **topsecret**. Esto se pensó así para poder agregar llamadas a los satélites desde la nave, pero siempre teniendo en cuenta que, para poder calcular la localización de la nave, estos satélites **no pueden ser más de 3 en total**.

Por lo dicho anteriormente, si en el agregado de distancias y mensajes de los satélites, se ingresa información de un satélite que ya existe en la lista, este mismo se actualizara con los nuevos datos (distancia y mensaje enviado por la nave).

Solo en caso que la nueva información del satélite ingresado, con nombre enviado a través de los parms de la url, no exista en la lista, entonces si se agregara como una nueva llamada de la nave. **Repitiendo este proceso hasta llegar a 3 el largo de la lista mencionada**.

Aclaración: este diseñó y desarrollo se llevó adelante de esta forma ya que el enunciado no dice nada sobre si el satélite debe agregarse a los ya existentes, o si la lista es nueva para este endpoint. Por lo que quedó a consideración del desarrollador.

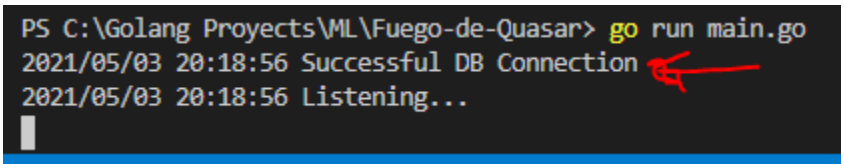
Database connection:

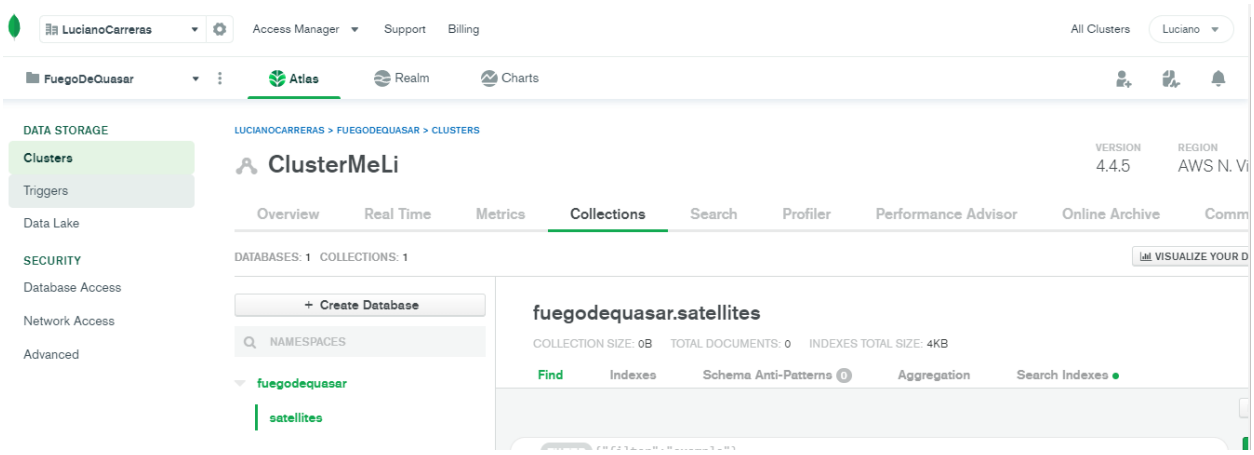
Por otra parte, se utiliza una conexión a un cluster de Base de datos en **MongoDB**. Si se logra dar con el cliente y se recibe respuesta, la API puede continuar.

Esto es para lo único que se utilizó la Base de Datos en este challenge. Solo para mostrar como conectarla, usarla, y hacerle un ping. Pero no se persistieron datos en la misma ni tampoco se obtuvieron datos.

Se consideró que, al trabajar solo **hasta** 3 satélites (para el cálculo de coordenadas de la nave), no era necesario para este challenge.

Por supuesto que, de haber tenido más tiempo, se podría haber implementado un repositorio para los datos manipulados.





Heroku:

Para poder subir la API a la nube, se utilizó HEROKU APP, que al igual que GIT utiliza un **repositorio** para subir y actualizar los archivos.

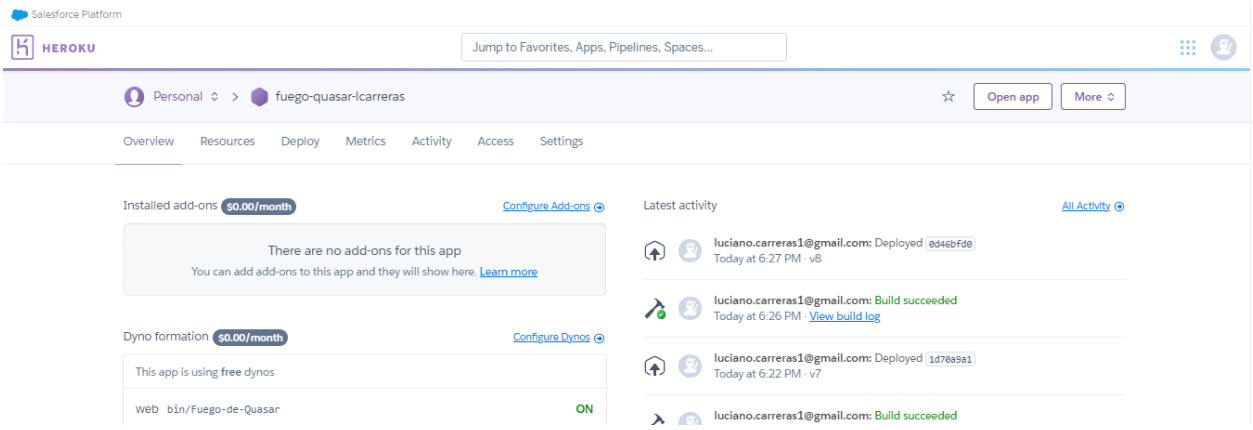
Al realizar los siguientes pasos, luego Heroku puede compilar y deployar los archivos de GO para hacer visible a la API.

git add .

git commit –m “ “

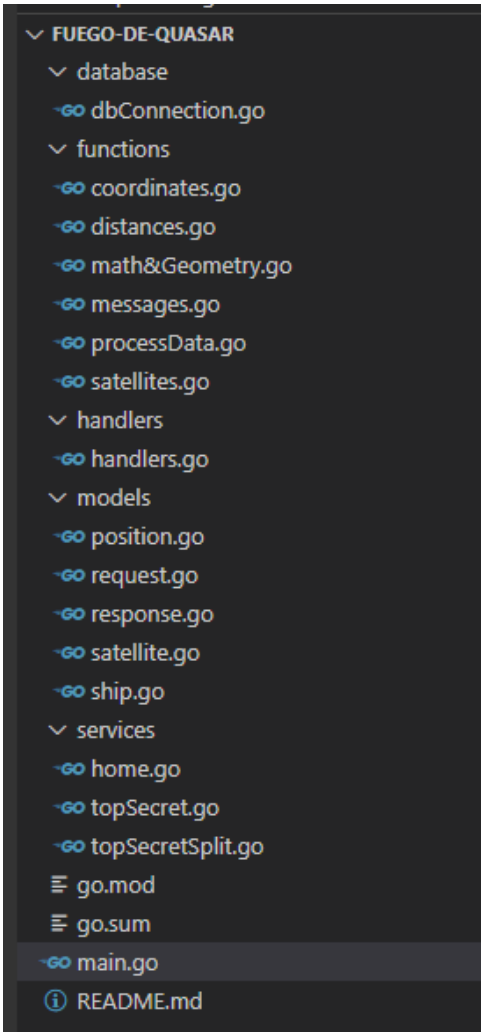
git push

git push heroku (utilizando solo la rama principal)



Estructura de archivos y carpetas:

A continuación, se muestra la **estructura** creada y utilizada desde Visual Studio Code:



Database:

En la carpeta database, tenemos el DBConnect para obtener el cliente de mongoDB. Ahí le pasamos un connection string para poder llegar al cluster creado. Luego hacemos pruebas de ping para verificar.

Handlers:

En la carpeta handlers definimos un router con **Gozilla Mux** y utilizamos **HandleFunc** para poder manejar los endpoints a través de funciones, llamadas a servicios, que dependiendo del método GET o POST al que se esté "pegando", vamos a proceder de una forma u otra en la API.

También definimos el Puerto de acceso a nuestra API, y los permisos de seguridad, que a través de CORS, los liberamos dando total acceso al router creado, y que dará lugar a los endpoints y sus HandleFunc.

Luego, con todo esto, ya podemos escuchar y servir desde el Puerto y el manejador indicado.

Services:

En la carpeta services definimos los 3 servicios a utilizar dependiendo del endpoint al que se quiera acceder desde la API. El servicio Home es para el endpoint "/", el servicio TopSecret es para el endpoint "/topsecret", y los servicios GetTopSecretSplit y PostTopSecretSplit para los endpoint "/topsecret_split" y "/topsecret_split/{satellite_name}" respectivamente. Recordamos que estas rutas se definieron dentro de los manejadores para llamar a una función particular de la API.

Functions:

En la carpeta functions, tenemos todas las funciones y métodos necesarios para poder trabajar desde la API, obtener información, recorrerla, y hacer los cálculos matemáticos antes mencionados para el cálculo de coordenadas.

Models:

En la carpeta models se crean todas las estructuras para representar los modelos de datos a utilizar. Ya sea para los satelites, nave, el request, el response, o la posición de la nave con sus coordenadas.

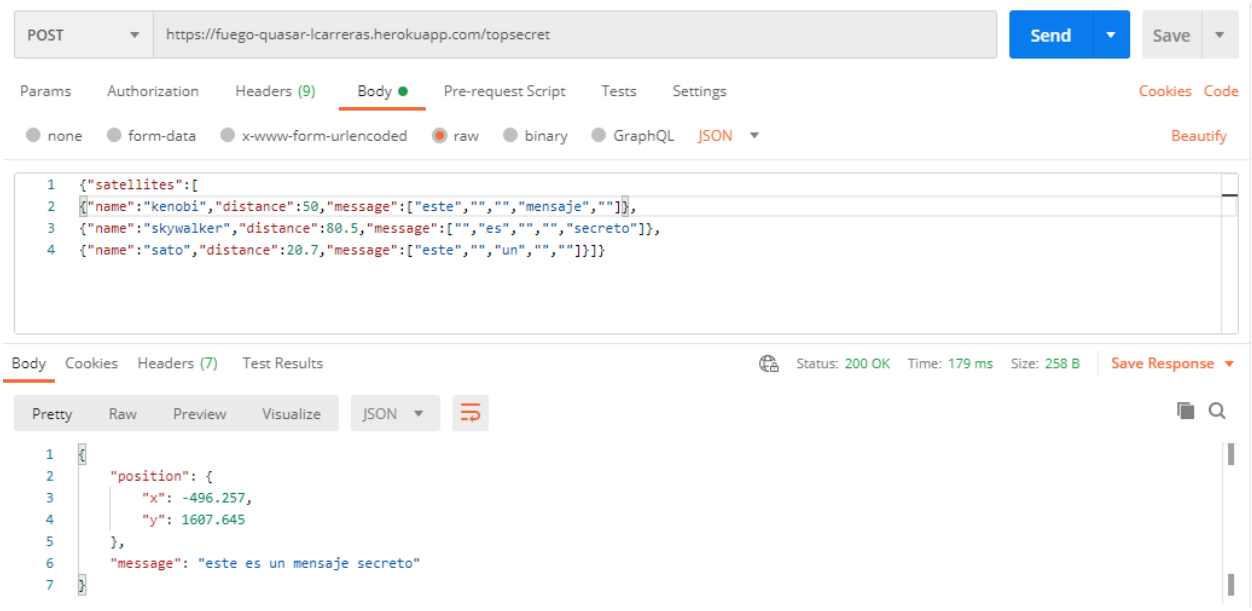
Main:

El archivo main.go, donde se encuentra la función main() que es la función principal para iniciar la API donde evaluamos si la base de datos de mongoDB esta accesible. Si es así, llamamos a la función Handlers para ubicar los endpoint con sus respectivos servicios, ya comentado anteriormente.

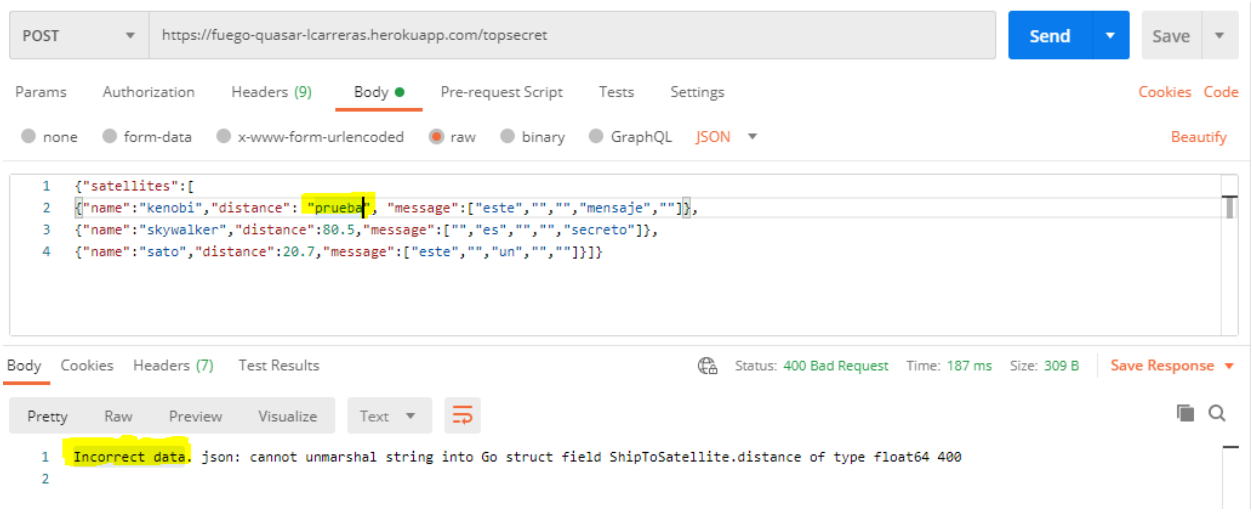
Luego se encuentran los archivos go.mod y go.sum, propios del lenguaje.

Pruebas realizadas desde Postman:

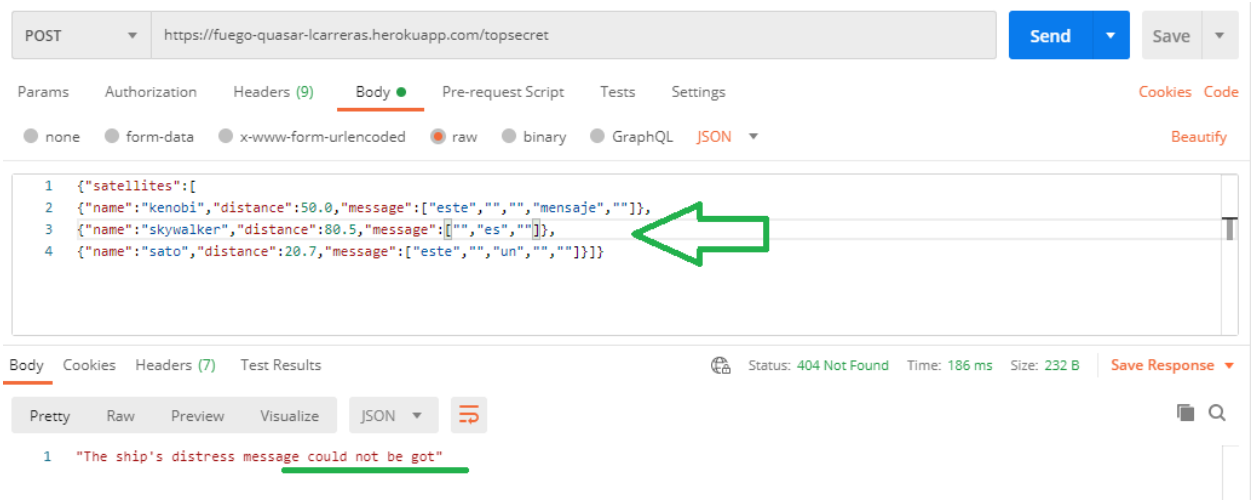
A TopSecret:



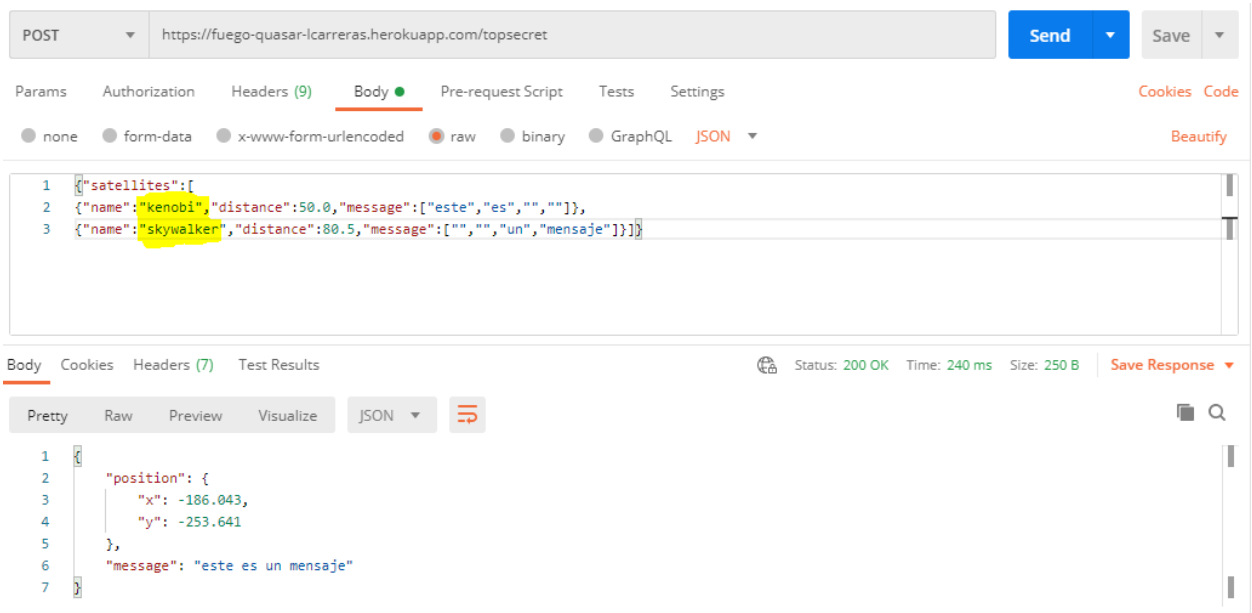
A TopSecret: con valor incorrecto para una distancia.



A TopSecret: con valor incorrecto o incompleto para el mensaje del satélite skywalker.

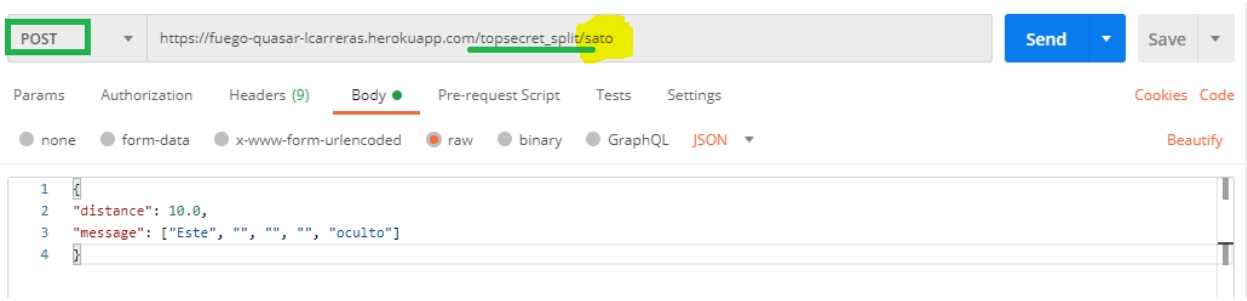


A TopSecret: distancia y mensajes de 2 satélites únicamente.

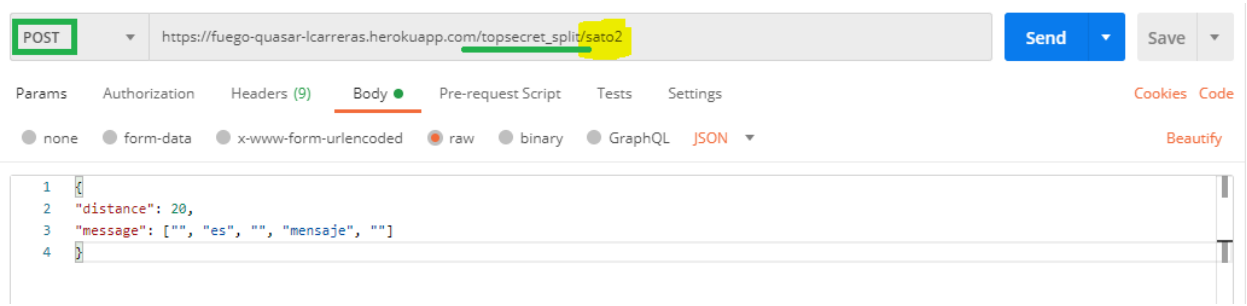


A TopSecret _split: agregamos el 1er satélite (recordando que se toma como incial que no hay satelites cargados – ver documentación sobre el punto 3 – es decir que para poder hacer un GET para recibir la información con las coordenadas y el mensaje, se deben cargar los satelites mediante un POST).

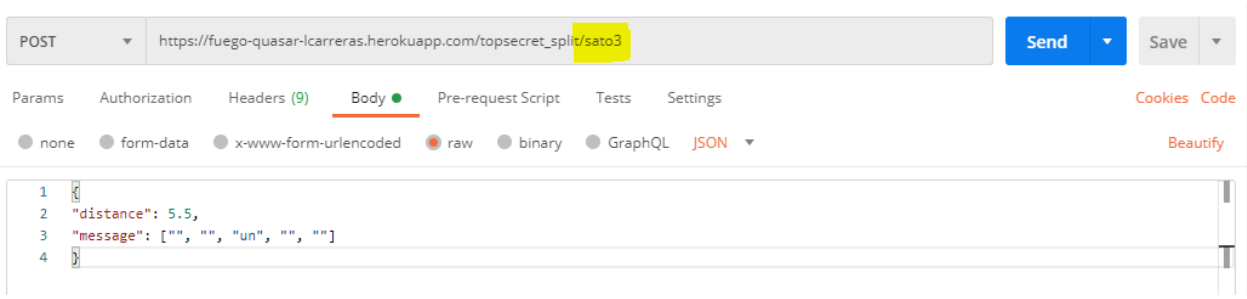
Agregamos el 1er satélite con sus datos:



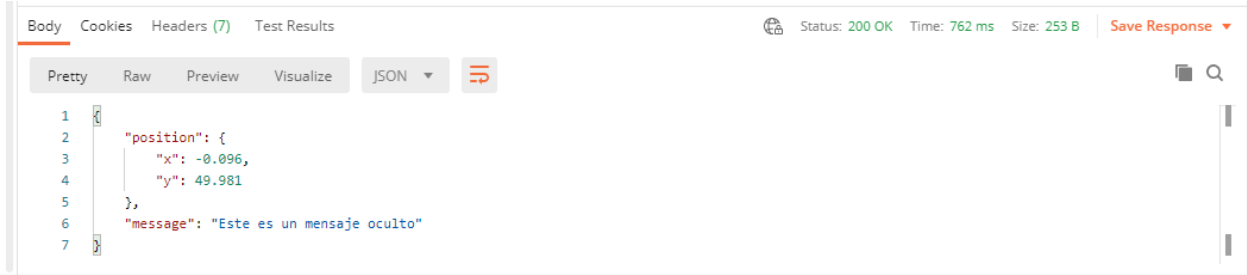
Agregamos un 2do satélite con sus datos:



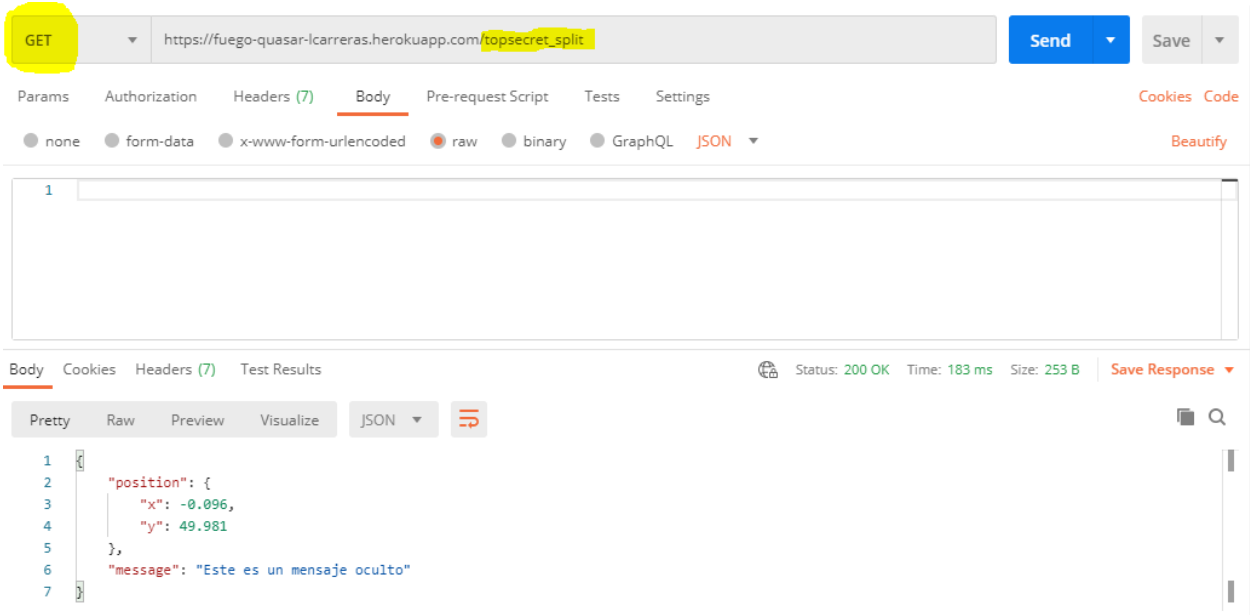
Agregamos un 3er satélite con sus datos:



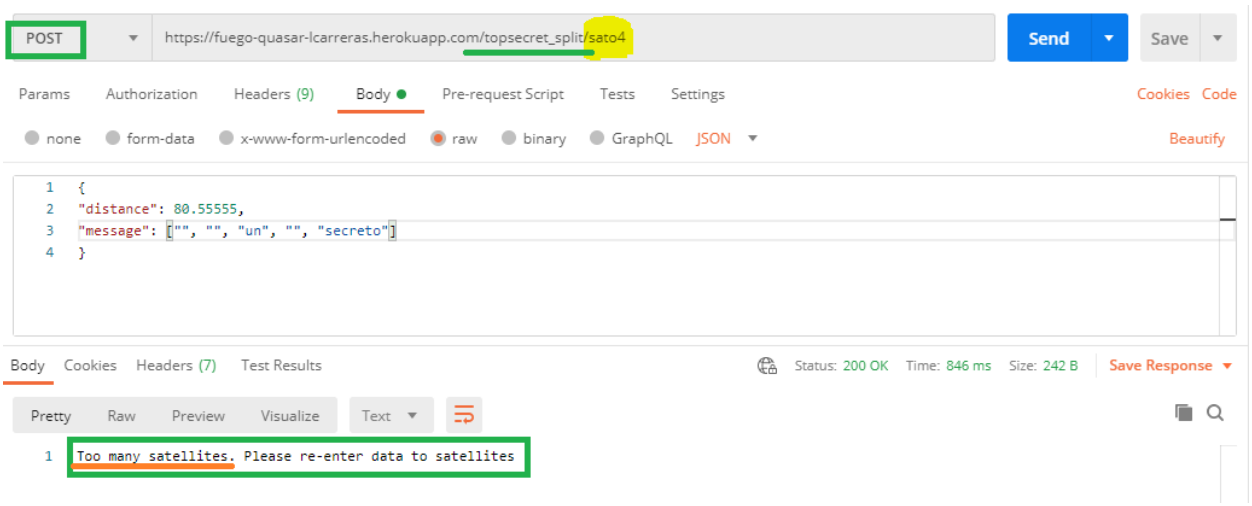
Como la data que se cargó es suficiente y esta correcta, vemos en el response el siguiente resultado:



Mismo resultado obtenemos, si en vez de hacer un POST de un satélite, hacemos un GET a “topsecret_split”:



Como tenemos la restricción de que solo podemos calcular la ubicación dados 3 satelites como máximo (ver documentación), si queremos añadir un satétlite mas, obtenemos como respuesta el mensaje de limitación, con el status OK (200).



Conclusiones:

Al no haber trabajado anteriormente con Golang, tuve que tomarme unos días para poder estudiar el lenguaje, sobre todo, sus características más importantes para conocer todo su potencial como los packages, Gorutinas, Interfaces, Mapas, channels, y por supuesto todo lo que se refiere a las API Rest.

Si bien había escuchado acerca del lenguaje de programación de Google, nunca había tenido la oportunidad de codificar y ver resultados en el mismo. Por lo que de alguna u otra manera este challenge me servirá para futuro. Me gustaría seguir aprendiendo y perfeccionando las metodológicas, patrones, etc. Ojalá sea en MeLi.

Agradezco la oportunidad de que me hayan conocido y espero hacerles de ayuda en algún momento.