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: fuego-quasar-lcarreras https://git.heroku.com/fuego-quasar-lcarreras.git

Aclaraciones y comentarios a tener en cuenta:

Para la function GetLocation 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 eran 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.

Para la función GetMessage, se tuvo en cuenta un desfasaje “hacia la derecha”. Es decir, a partir de encontrar el slice de strings más corto, se analizaba con el largo de los slice restantes. Si el largo de un mensaje era mayor, se considera a analizar las palabras a partir de la ubicación “i + desfasaje”. Es decir, si el mensaje 1 tenía un largo de 4 palabras, y el largo mínimo encontrado es de 3, al mensaje 1 se comienza a analizar su contenido a partir de la posición 1, y no desde la posición 0, dado el desfasaje calculado por el desarrollador.

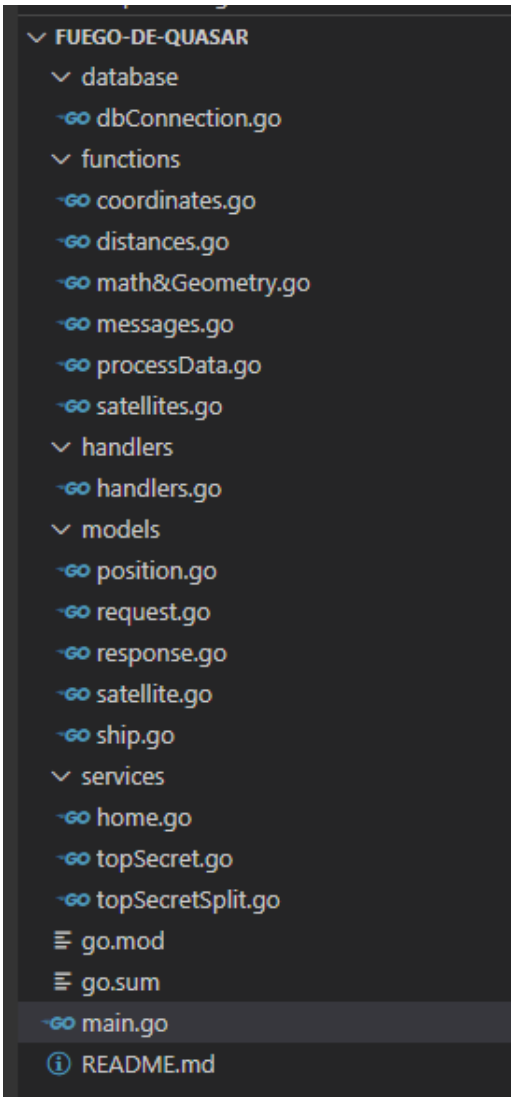
Para el servicio topsecret_split se tuvo en cuenta que el listado de llamadas a los satelites arranca vacia. Es decir que NO es la misma que para el servicio topsecret. Esto se penso asi para poder agregar llamadas a los satélites desde la nave, pero siempre sabiendo que para poder calcular la localizacion de la nave, estos satélites no pueden ser más de 3 en total. Es decir que si en el agregado de distancias a los satélites, 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 para el satélite, con nombre enviado a través de los parms de la url, no exista en la nueva lista, entonces ahí se agregara como una nueva llamada de la nave. Repitiendo este proceso hasta llegar a 3 el largo de la lista mencionada. Esto se pensó así ya que en el enunciado no dice nada si el satélite debe agregarse a los ya existentes o si la lista es nueva para este endpoint.

Por otra parte, se utiliza una conexión a un cluster de BD 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 BD, 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 poder solo trabajar a partir de 3 satélites, no era necesario para este challenge. Por supuesto que de haber tenido más tiempo, se podría haber implementado un repositorio para la data de la BD.

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 hacer estos pasos (git add ., git commit –m “ “, git push, y git push heroku, utilizando solo la rama principal y ninguna otra), luego Heroku puede compilar y deployar para hacer visible a la API.

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



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

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

En la carpeta handlers definimos un router con Gozilla Mux y utilizamos HandleFunc para poder manejar los endpoints a través de funciones, llamadas servicios, que dependiendo del método GET o Post que se este "pegando", vamos a proceder de una forma u otra en la API. También definimos el Puerto de acceso a nuestra API, y los permisos seguridad, que a traves de CORS, los liberamos y damos total acceso al router que dará lugar a los endpoints y sus handles func.

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

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.

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.

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

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