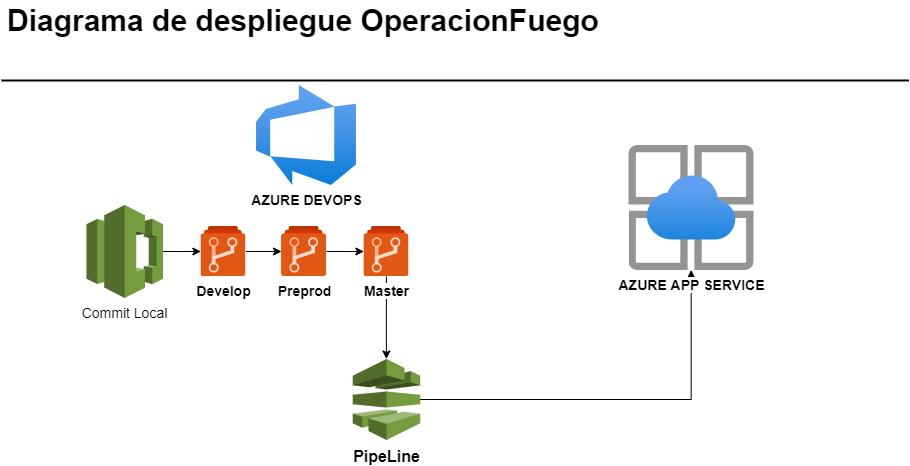
**Diseño de solución para Operación Fuego de Quasar.**

Para cumplir con la necesidad del servicio se construyó una solución en Golang orientada a microservicios la cual se encuentra diseñada para garantizar su operación 7X24.

El Diseño contempla los siguientes ítems:

* Alta disponibilidad.
* Protocolo https.
* Seguridad de acceso bajo Autenticación de AZURE APIM.
* Escalabilidad Horizontal.
* Escalabilidad Vertical.
* Documentación Swagger.
* Control de código fuente.
* Despliegue continuo.

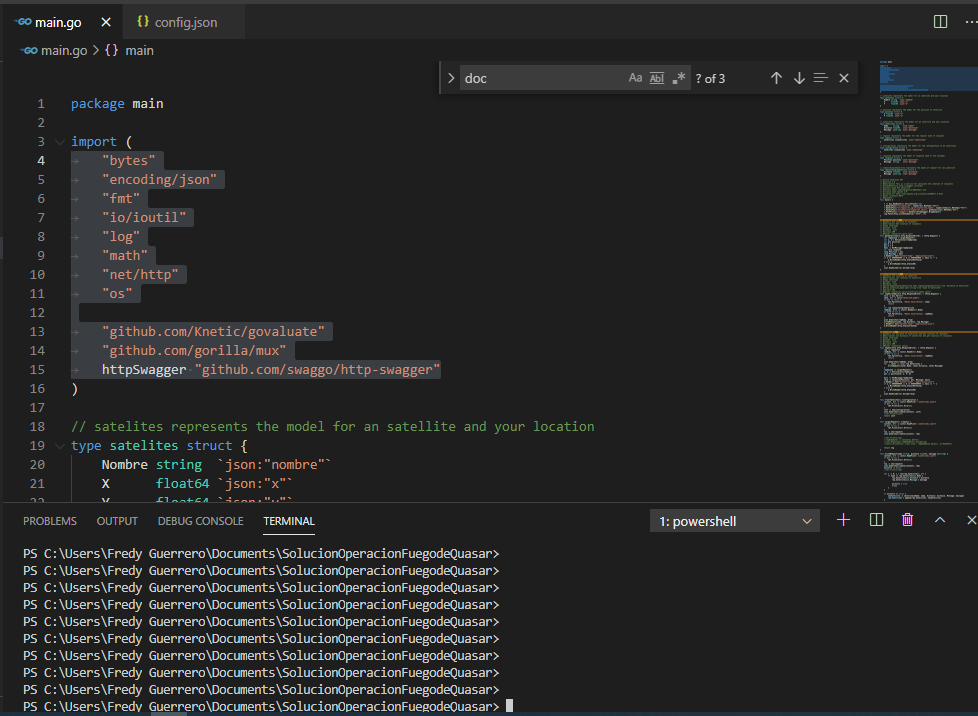
Diseño de la solución:



**Instrucciones de ejecución de la aplicación**

Como prerequisito se requiere haber instalado previamente Golang en el computador donde se ejecutará el programa.

Se debe abrir el visual studio code o la herramienta de desarrollo que use y abrir la carpeta en donde descargo la solución, abrir la consola de comandos o el terminal de visual studio code.



Ahora debemos importar los módulos que son necesarios para el funcionamiento de la aplicación.

    github.com/Knetic/govaluate

    github.com/gorilla/mux

    github.com/swaggo/http-swagger

github.com/githubnemo/CompileDaemon

github.com/swaggo/files

github.com/swaggo/swag/cmd/swag

github.com/Microsoft/ApplicationInsights-Go/appinsights

se debe ejecutar el comando “go get -u” agregándole las rutas de importación.

Ejemplo go get -u github.com/swaggo/swag/cmd/swag

para generar la carpeta de documentación del swagger se debe correr el comando.

swag init -g main.go

configure en el archivo main.go el puerto deseado para levantar el servicio, en la siguiente linea.

log.Fatal(http.ListenAndServe(":9876", r))

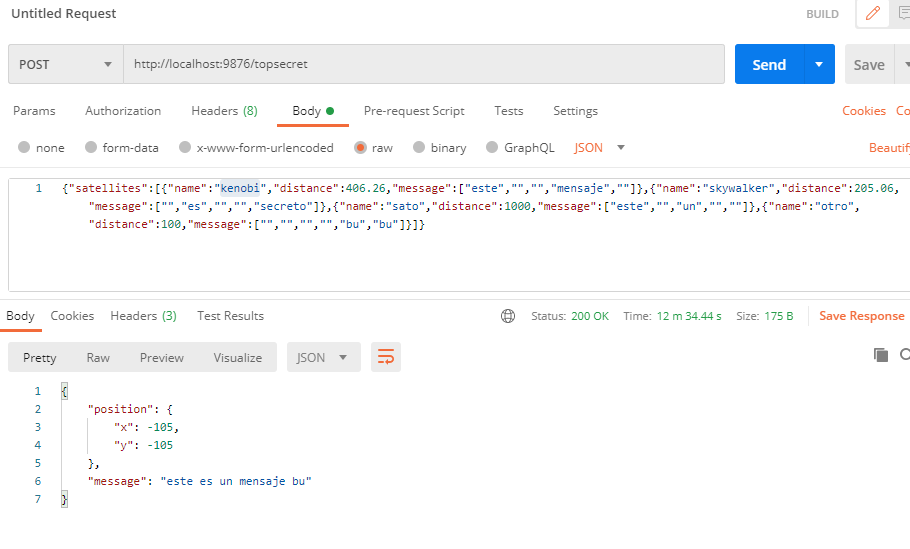
Ahora se puede correr el comando para lanzar el programa ya que hemos importado el compileDaemon se corre el siguiente comando para que el servicio se publique y poder editarlo en tiempo real visualizando los cambios al guardar.

Se debe crear el .exe con el comando CompileDaemon

Después se puede correr el comando para que en tiempo de ejecución se reinicie el servicio luego de cada cambio y lo despliegue.

CompileDaemon -command="SolucionOperacionFuegodeQuasar.exe"

Ahora se puede consumir los servicios desde el localhost usando una herramienta como postman.



**Configuración de la aplicación**

El archivo de la configuración config.json contiene los satélites con las ubicaciones y sus coordenadas este archivo permite agregar, modificar o eliminar satélites.

Ejemplo:

{"satelites": [{"nombre": "kenobi","x":-500, "y":-200}, {"nombre": "skywalker","x":100, "y":-100}, {"nombre": "sato","x":500, "y":100}]}

En caso de requerir agregar más satélites solo se deben incluir en el array satelites un nuevo item.

**Información Funcional de la aplicación**

Para cumplir con la necesidad de ejecutar un proceso que calcule las coordenadas de la nave que envié el mensaje realice dos soluciones que se pueden ver en el código fuente.

1. ***Despeje de Formula Distancia.*** La primera solución es utilizar la formula de distancia en donde luego de despejarla usando dos satélites se obtiene una ecuación cuadrática en donde se calcula la coordenada x dándonos 2 valores que se deben comprobar contra la fórmula de distancia para identificar el valor correcto, luego se usa la coordenada x para calcular la y obteniendo finalmente las coordenadas, esta fórmula es tan compleja que al realizar el despeje dio una ecuación demasiado grande que no fue posible programarla, adicional este proceso representa un problema ya que al aplicar las fórmulas a causa de los decimales en el redondeo puede generar una desviación que no permite calcular exactamente los valores y al comprobarlos en la fórmula de distancia original no da el resultado correcto, (Anexo proyecto java donde esta las fórmulas que me dieron en el despeje y me gustaría que me compartieran la formula despejada para ver que error pude cometer a despejar).
2. ***Calculo con matriz de opciones.*** Esta segunda opción consiste en trazar una matriz para recorrer los posibles valores y así calcular por comprobación sobre la aplicación de la formula de distancia sobre todos los satélites las coordenadas correctas, el problema de esta solución es el tiempo de procesamiento ya que requiere de mucho tiempo y consume los recursos de la máquina sin embargo es la opción que finalmente implemente debido a que esta funciona correctamente, aunque demora en responder, **NOTA**: Provisionalmente la restricción es para la coordenada X desde el punto -110 hasta el 200, para la coordenada Y desde el -110 hasta el 200.

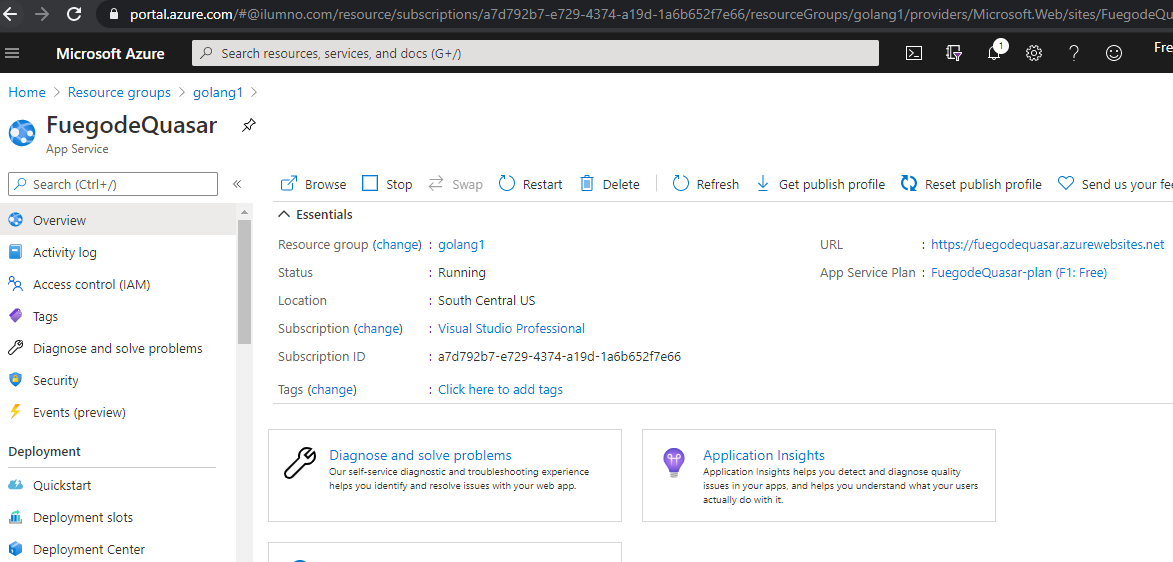
con esto es posible realizar el cálculo para el siguiente body.

{"satellites":[{"name":"kenobi","distance":406.26,"message":["este","","","mensaje",""]},{"name":"skywalker","distance":205.06,"message":["","es","","","secreto"]}}]}

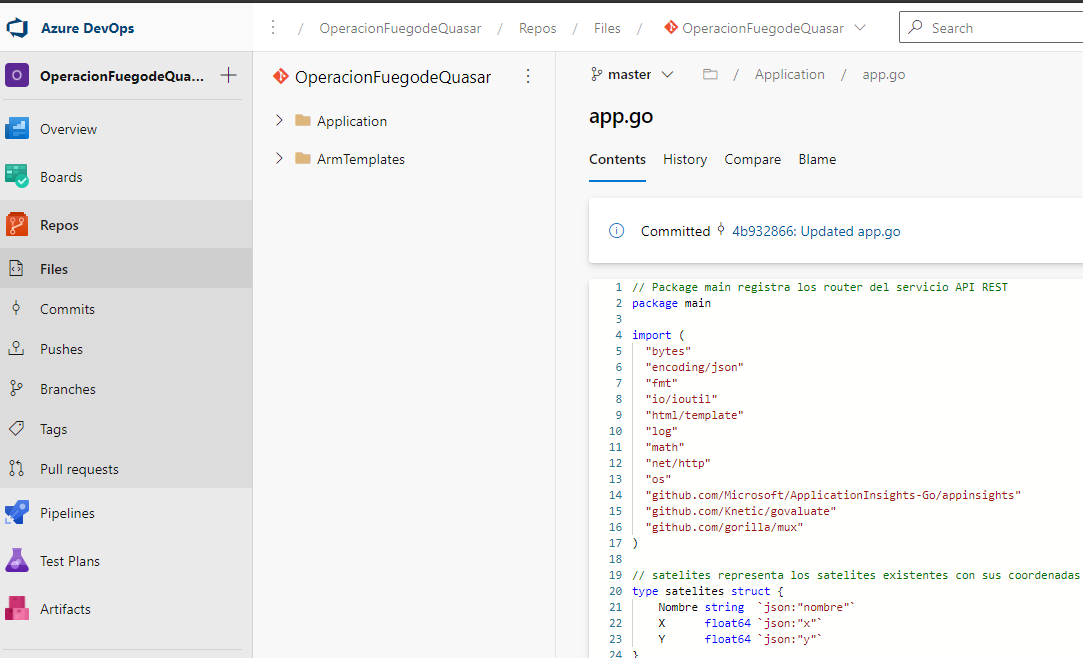
En relación con el cálculo del mensaje, se utilizo una fórmula de comprobación que consiste en validar todos los mensajes tomando como campo el valor que coinciden en su misma posición del array en donde llegan los mensajes, de esta forma teniendo en cuenta que se crea una matriz los valores que coincida o los que estén vacíos permiten identificar cuando un texto del mensaje es correcto y se excluyen los que tienen múltiples valores ya que no permiten decidir cuál es correcto.

**Información General de la solución.**

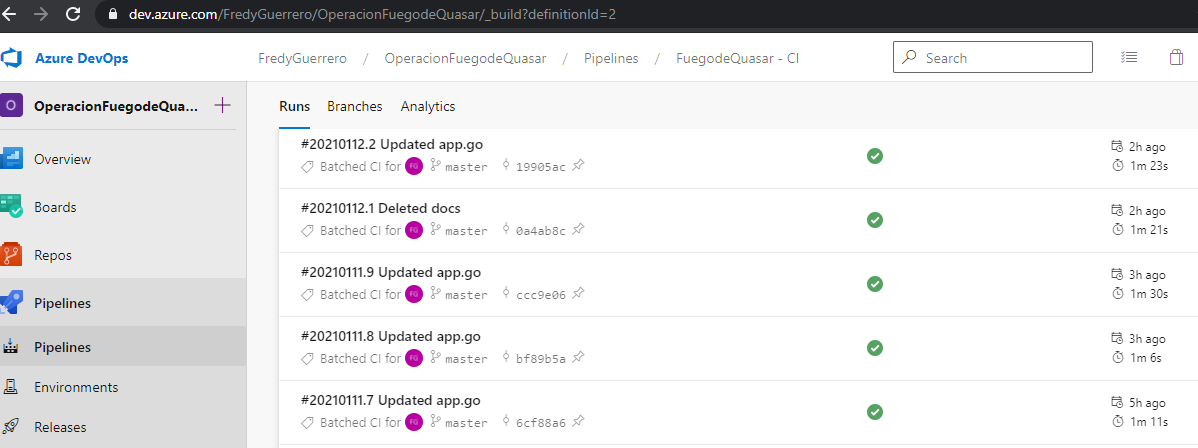
Para publicar la solución se creó en AZURE DEVOPS un APP SERVICE quedando de la siguiente forma:



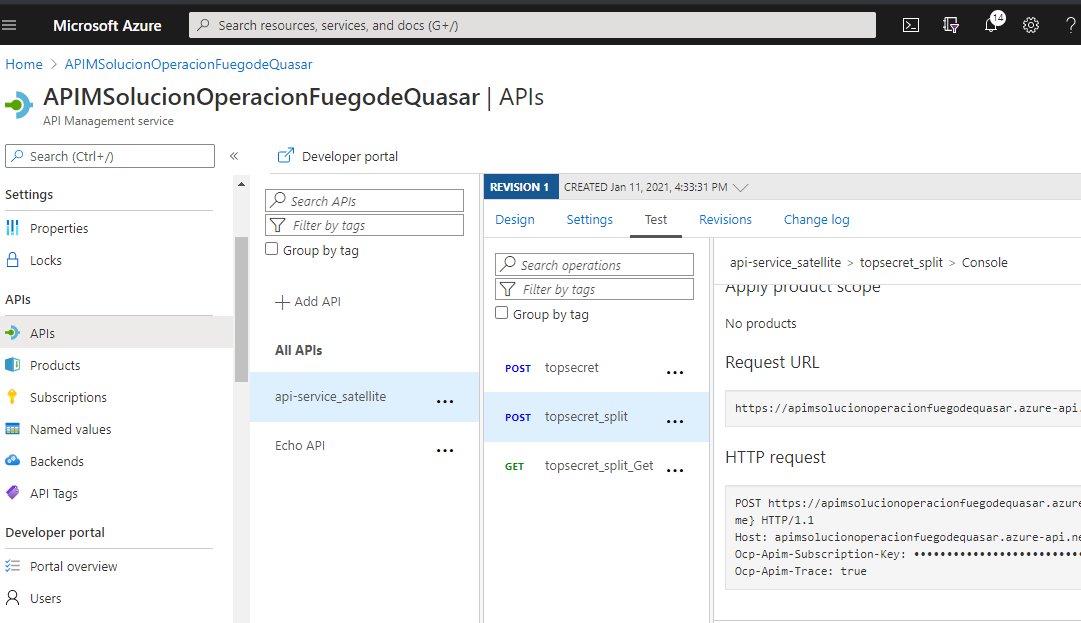
Dentro del Azure devops se realiza el control de código fuente y despliegue de las versiones hacia la rama master para que sea publicado en producción.



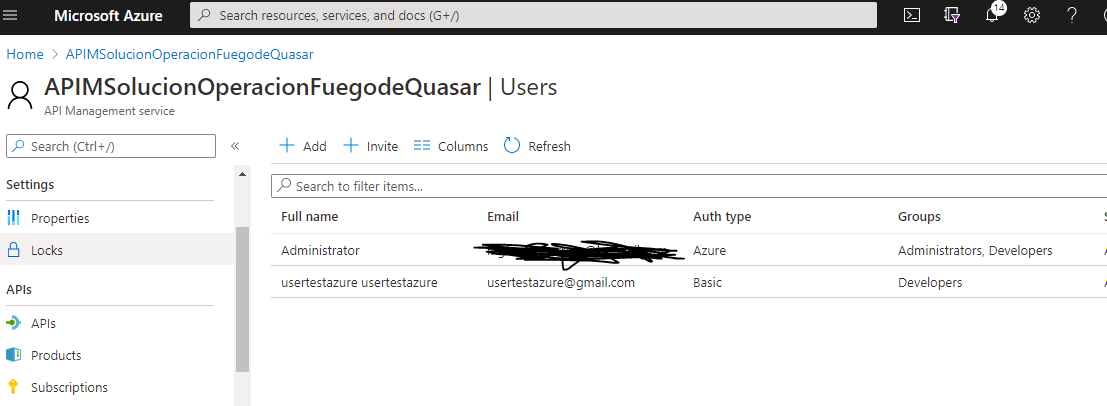
Los PipeLines se encargar del despliegue continuo de la aplicación.



Se creo un APIM MANAGER en AZURE.



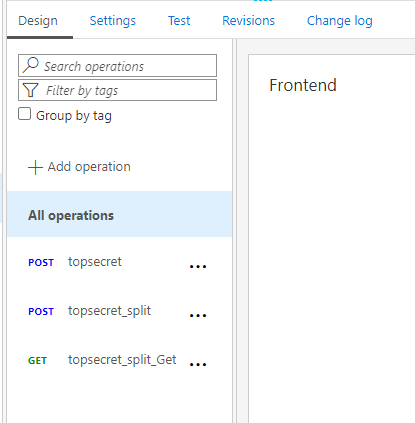
Se creo el usuario de consumo



Se creo una suscripción para autenticación de la aplicación.



Finalmente, se desplego la API con sus operaciones.



**URL Repositorio GitHub**

[**https://github.com/ingfredyguerrero/OperacionFuegodeQuasar**](https://github.com/ingfredyguerrero/OperacionFuegodeQuasar)

**URL Publicación Producción**

<https://operacionfuegodequasar.azure-api.net>

Autenticación por suscripción

Llave para autenticarse

'Ocp-Apim-Subscription-Key: 1a2c13f7071542d39067d3157d893da1'

**Ejemplos de consumo**

Curl para POST topsecret

curl --location --request POST 'https://operacionfuegodequasar.azure-api.net/v1/topsecret' \

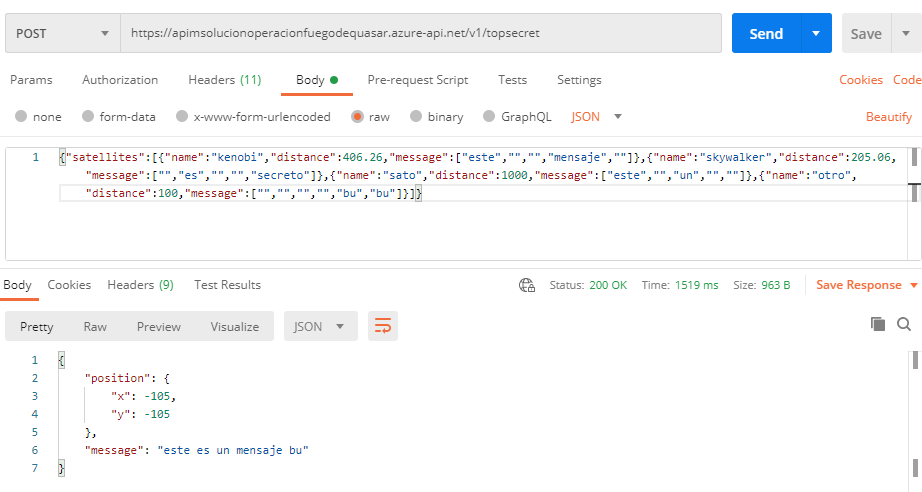
--header 'Ocp-Apim-Subscription-Key: 1a2c13f7071542d39067d3157d893da1' \

--header 'Ocp-Apim-Trace: true' \

--header 'Content-Type: application/json' \

--data-raw '{"satellites":[{"name":"kenobi","distance":406.26,"message":["este","","","mensaje",""]},{"name":"skywalker","distance":205.06,"message":["","es","","","secreto"]},{"name":"sato","distance":1000,"message":["este","","un","",""]},{"name":"otro","distance":100,"message":["","","","","bu","bu"]}]}'

Evidencias.



Curl para POST topsecret\_split/{satellite\_name}

curl --location --request POST 'https://operacionfuegodequasar.azure-api.net/v1/topsecret\_split/kenobi ' \

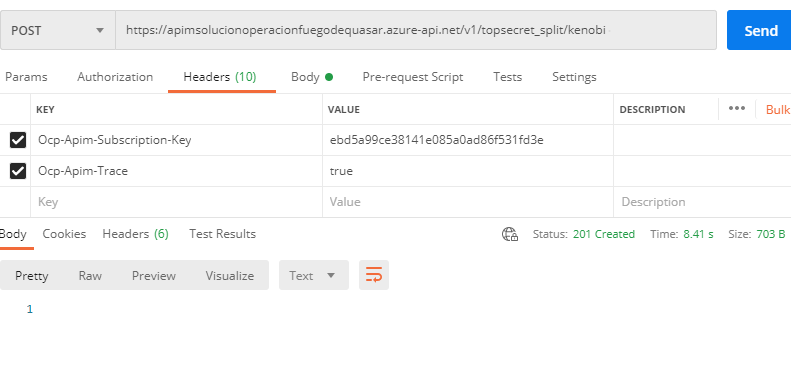
--header 'Ocp-Apim-Subscription-Key: 1a2c13f7071542d39067d3157d893da1' \

--header 'Ocp-Apim-Trace: true' \

--header 'Content-Type: application/json' \

--data-raw '{"distance":200,"message":["este","","un","","prueba4"]}'

Evidencias



Curl para GET satellite/topsecret\_split

curl --location --request GET 'https://operacionfuegodequasar.azure-api.net/v1/satellite/topsecret\_split' \

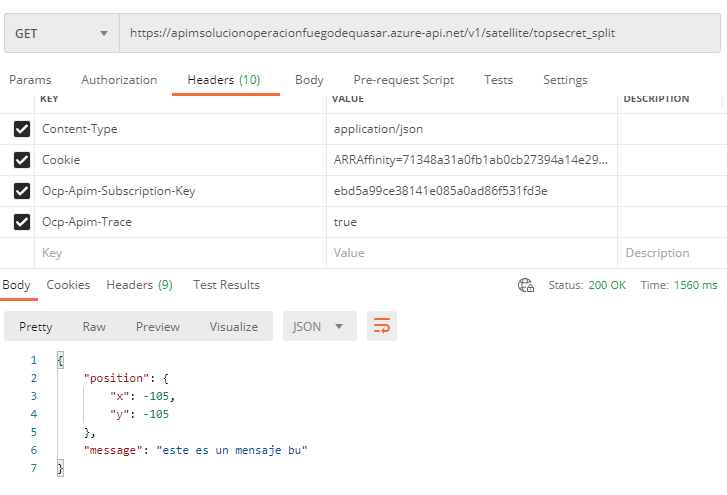
--header 'Content-Type: application/json' \

--header 'Cookie: ARRAffinity=71348a31a0fb1ab0cb27394a14e292875f39c36acbe2721cdaa782e123565e70; ARRAffinitySameSite=71348a31a0fb1ab0cb27394a14e292875f39c36acbe2721cdaa782e123565e70' \

--header 'Ocp-Apim-Subscription-Key: 1a2c13f7071542d39067d3157d893da1' \

--header 'Ocp-Apim-Trace: true'

Evidencias



**Buenas practicas**

Como buenas prácticas se implementaron a nivel de codificación las recomendaciones que se muestran en la página principal de GOLAND.

* Manejo de errores.
* Documentación de código.
* Definiciones de variables.
* Reutilización de código.
* Menor cantidad de líneas para optimizar su funcionamiento.

