As you may have heard, Han Solo, the newly appointed General of the Rebel Alliance, has initiated Operation Quasar Fire. The objective? Deal a decisive blow to the oppressive Galactic Empire. Our mission is clear: intercept the Imperial cargo ship's auxiliary call and recover its crucial cargo: rations and weaponry destined for an entire legion of troops.

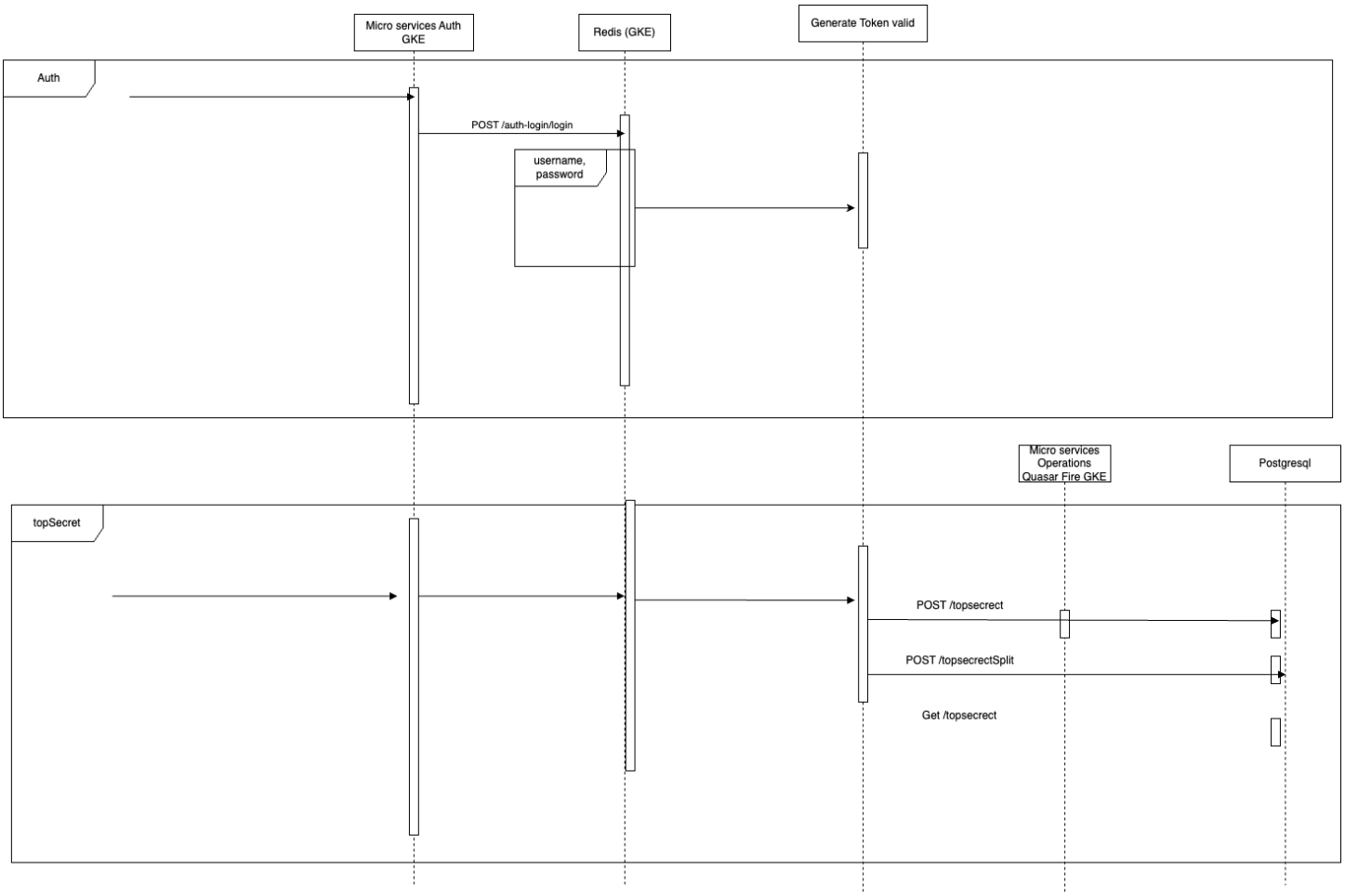
Step by Step guide to using API

This is a step by step to use the Operation Quasar Fire API:

- Choose the endpoint to use:
  - You must decide whether you want to use the CreateTopSecret or CreateTopSecretSplit endpoint depending on your needs.
- Get the authentication token:
  - Uses the authentication collection provided in the repository to obtain an authentication token.
  - Copy the generated token.
- Add the token to the Headers:
  - Make a call to the selected endpoint.
  - Add the token to the API header with the Authorization key and the Bearer value followed by the generated token.
- Choose the number and type of satellites:
  - If you chose CreateTopSecret, make sure to upload an array with the information of the satellites to decrypt. This array cannot contain more than 3 satellites and must include valid information.
  - If you selected CreateTopSecretSplit, add the satellites one by one. You cannot add more than 3 and they must be 3 different satellites.
- Create the satellites:
  - Once the satellites are selected and configured, use the corresponding endpoint to create them in the system.
  - In the case of CreateTopSecret, the satellites will be saved in a PostgreSQL database.
  - For CreateTopSecretSplit, satellites are added one by one and cached.
- Use the GetTopSecret endpoint:
  - This endpoint allows you to obtain the message information and the position of the ship using information from the satellites.
  - Provide the input parameter of the satellite you want to search for.
  - Note that this endpoint loads the information from a cache that was previously configured to determine how long it is saved.

By following these steps you will be able to use the Operation Quasar Fire API to determine the position and message of a ship in distress using information from satellites.

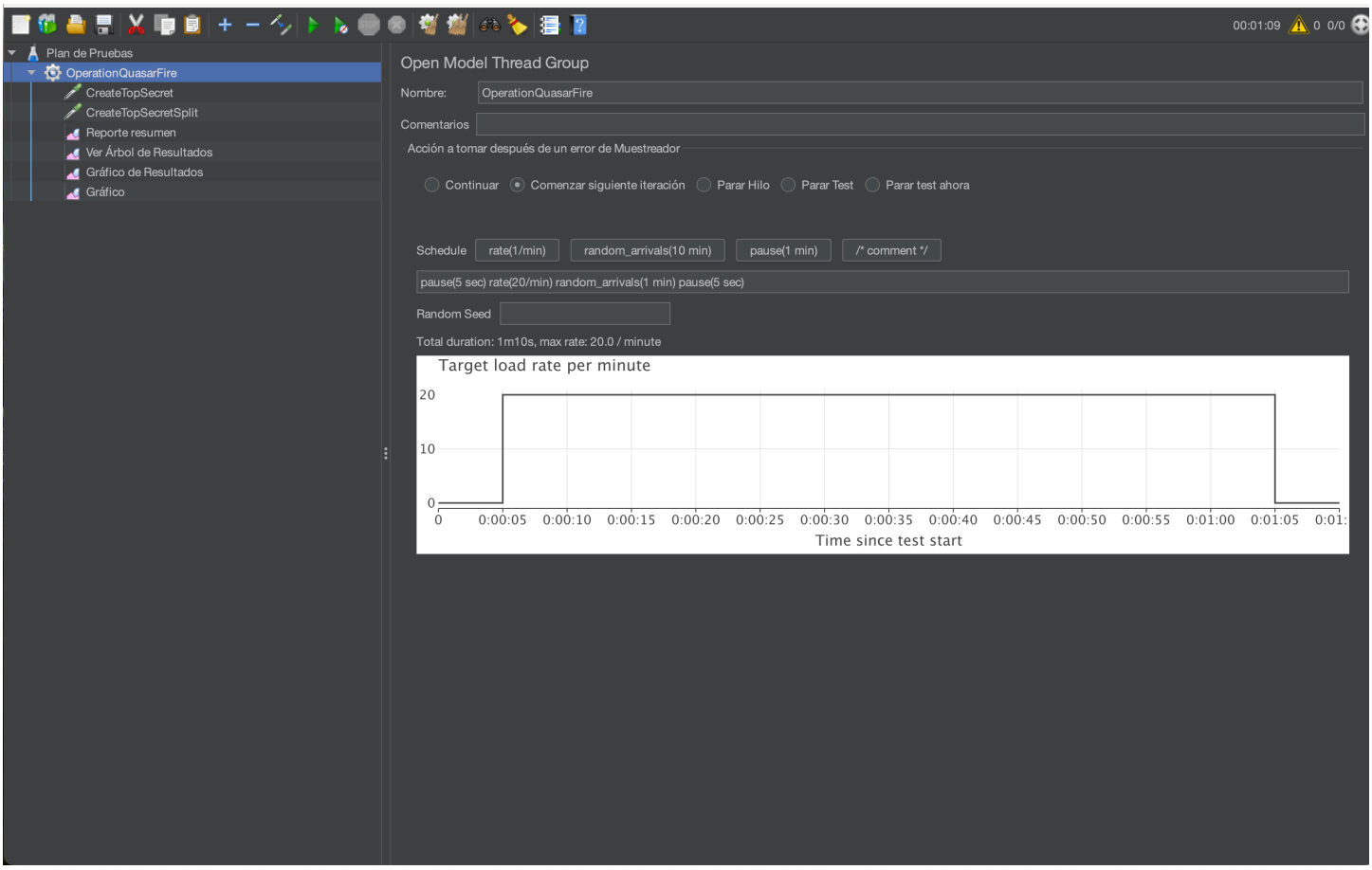
Proposed architecture



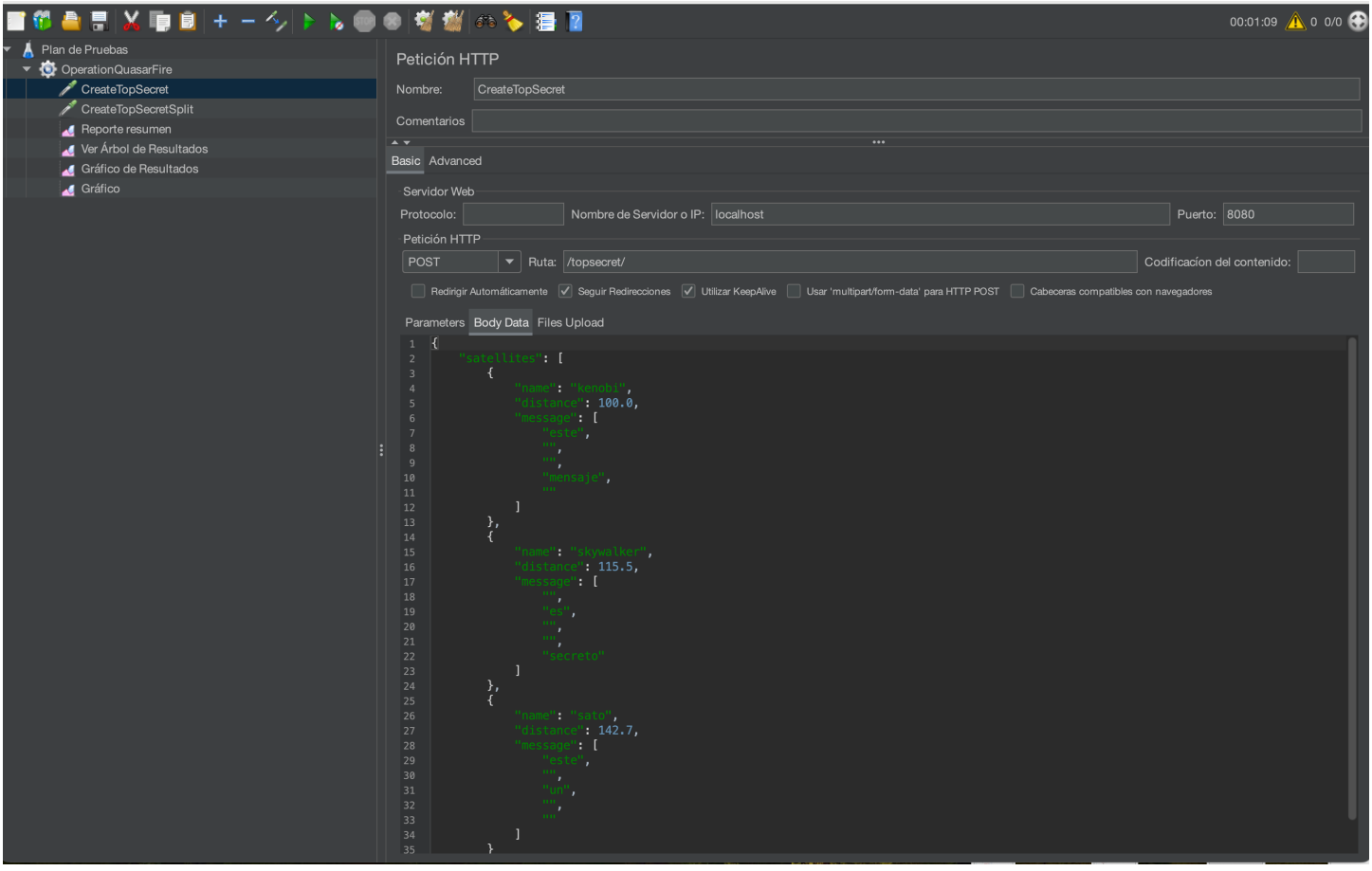
Documentation Load testing with JMETER

Below are the load model that were made for our OperationQuasarFire API

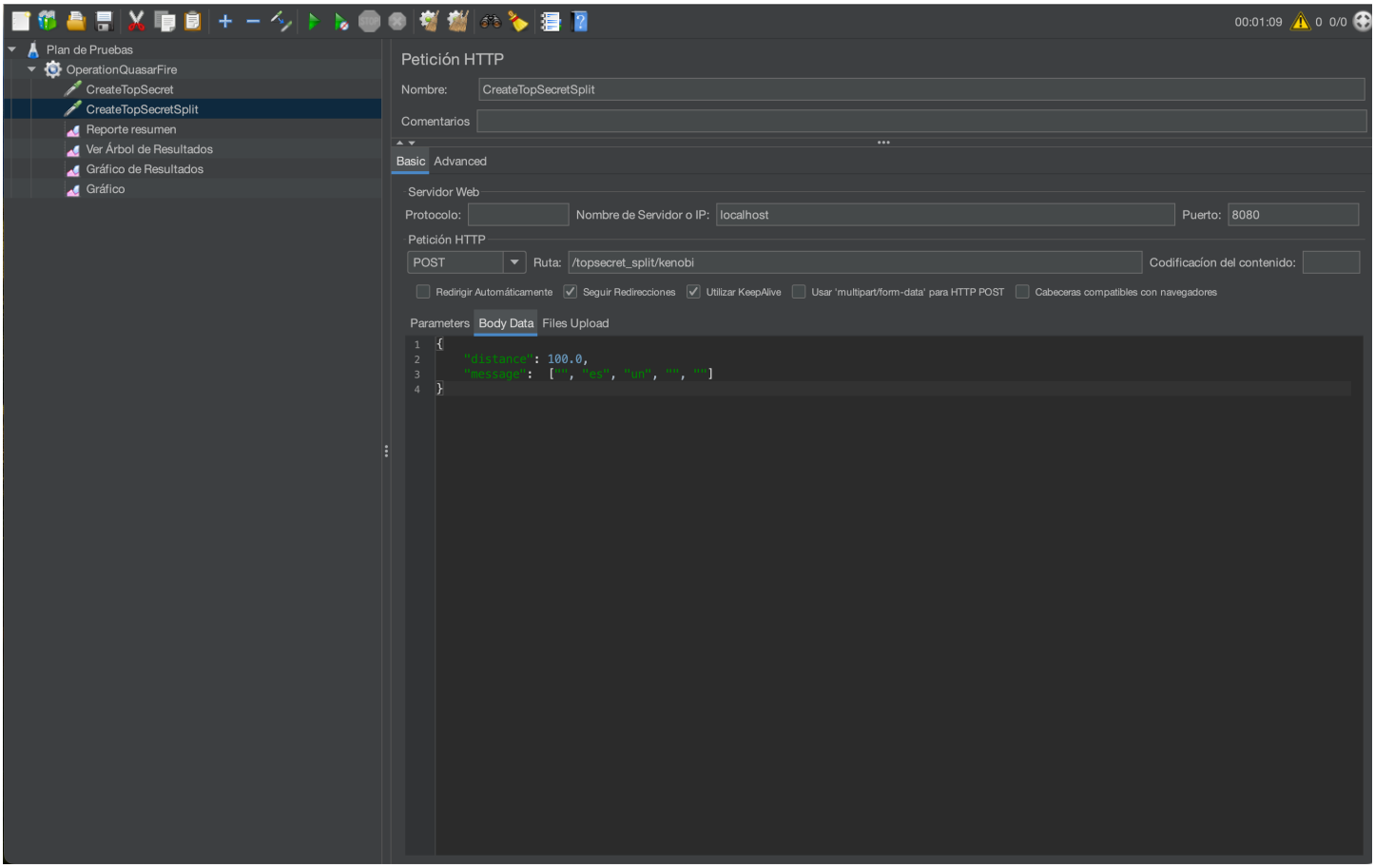
- Configuration



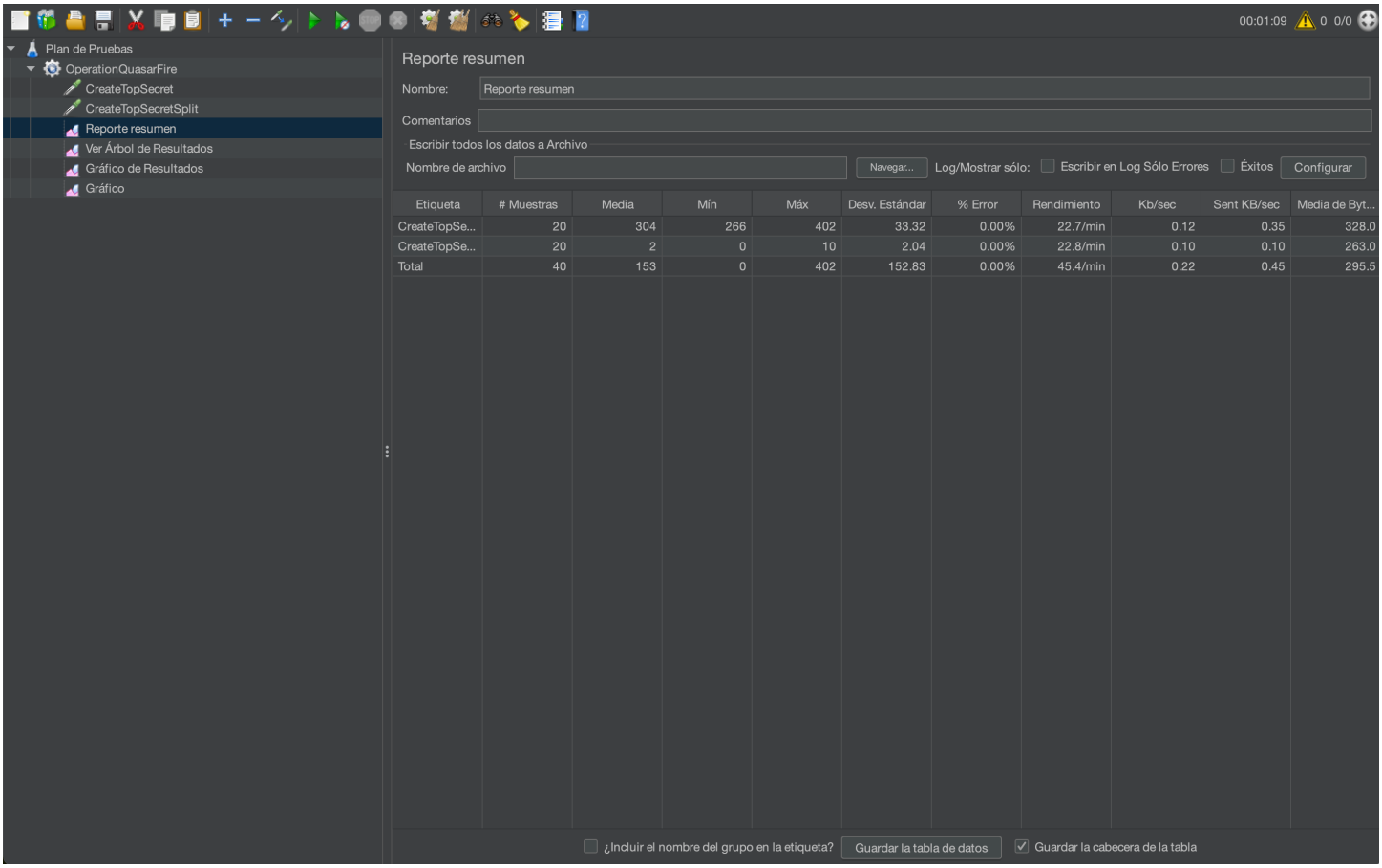
- CreateTopSecret

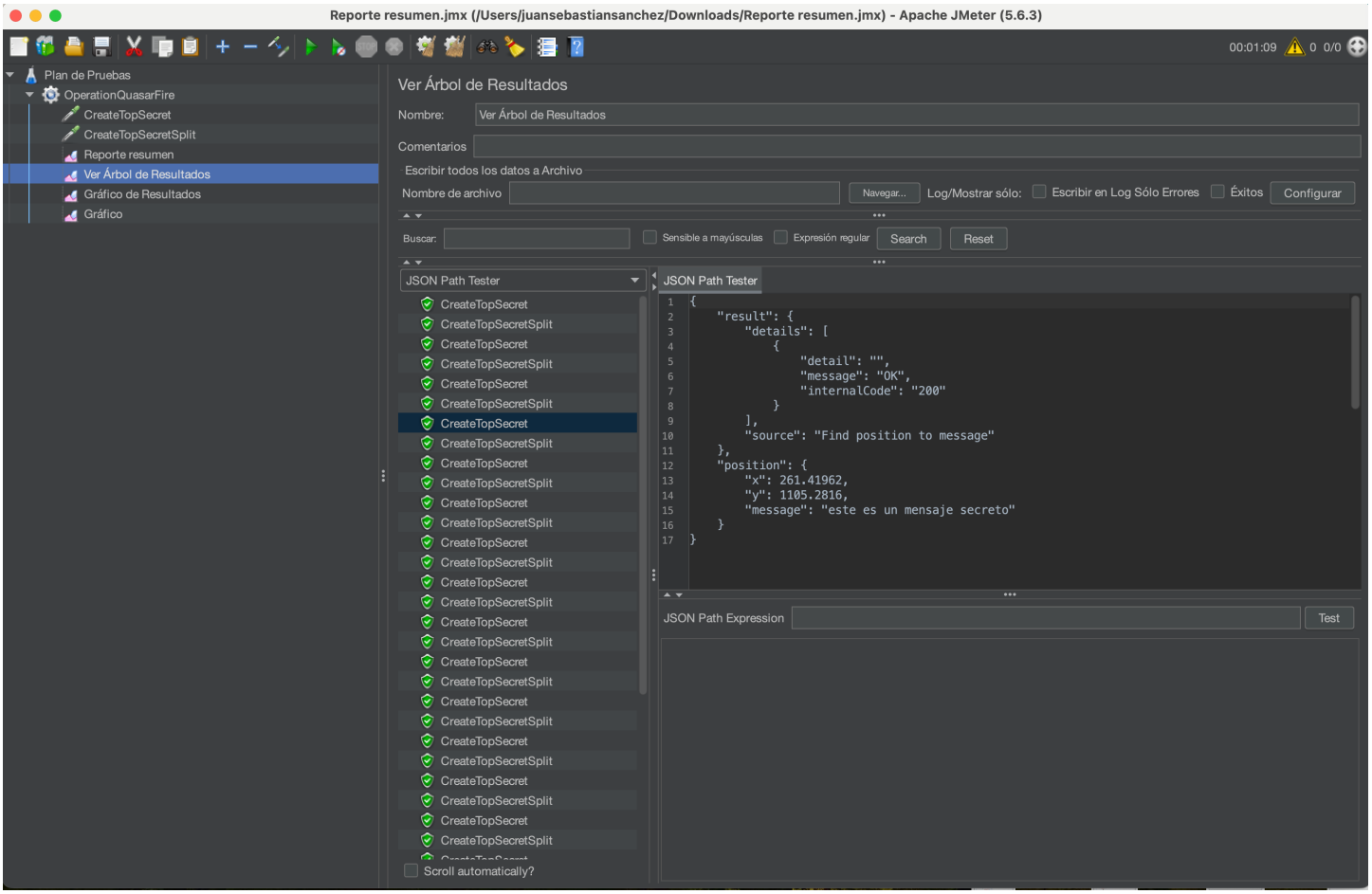


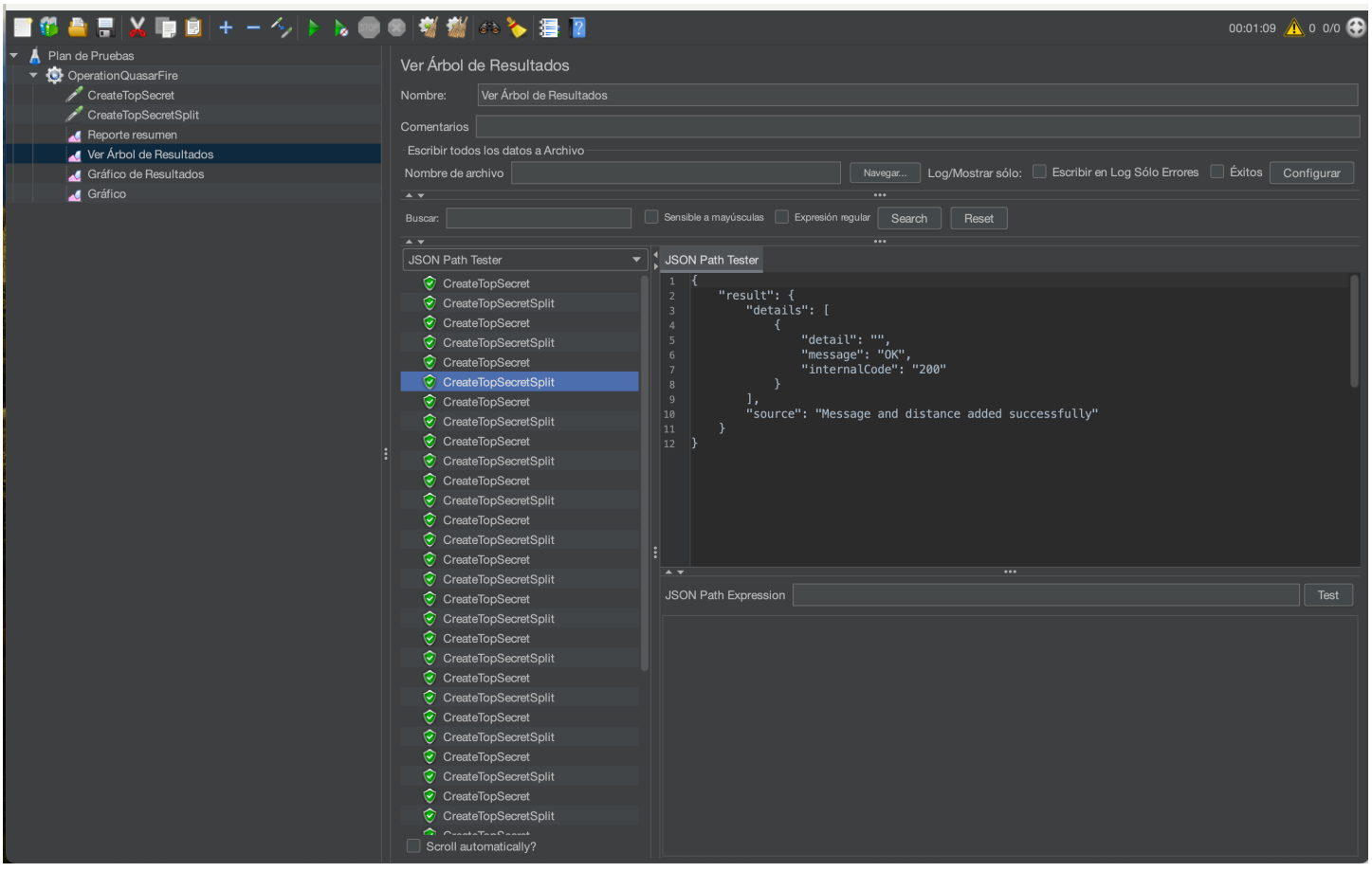
- CreateTopSecretSplit

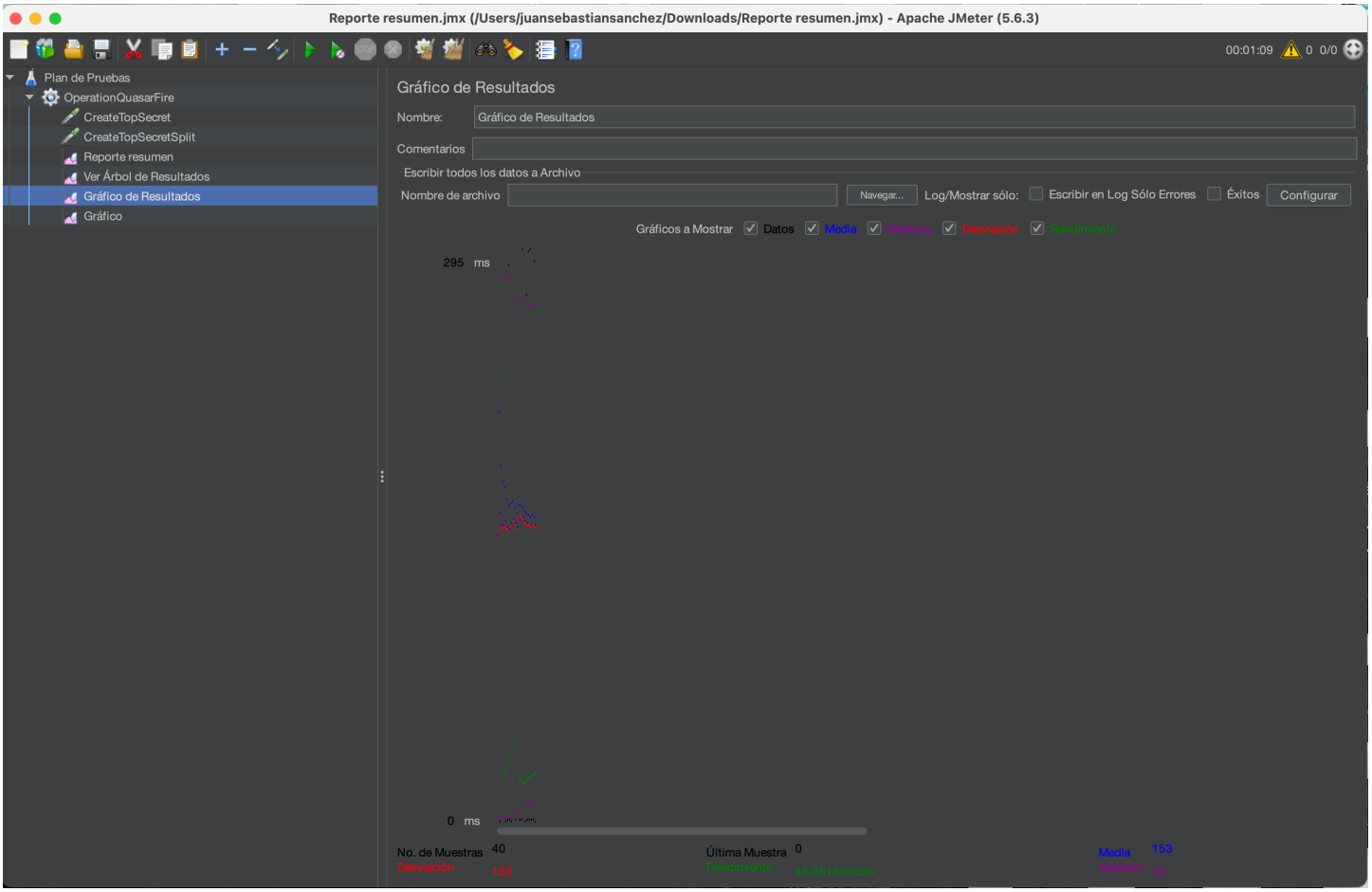

00:01:09 0 0/0
Plan de Pruebas
OperationQuasarFire
CreateTopSecret
CreateTopSecretSplit
Reporte resumen
Ver Árbol de Resultados
Gráfico de Resultados
Gráfico
Petición HTTP
Nombre: CreateTopSecretSplit
Comentarios
Basic Advanced
Servidor Web
Protocolo:
Nombre de Servidor o IP: localhost
Puerto: 8080
Petición HTTP
POST
Ruta: /topsecret_split/kenobi
Codificacíon del contenido:
Redirigir Automáticamente
Seguir Redirecciones
Utilizar KeepAlive
Usar 'multipart/form-data' para HTTP POST
Cabeceras compatibles con navegadores
Parameters Body Data Files Upload
{
"distance": 100.0,
"message": ["", "es", "un", "", ""]
}

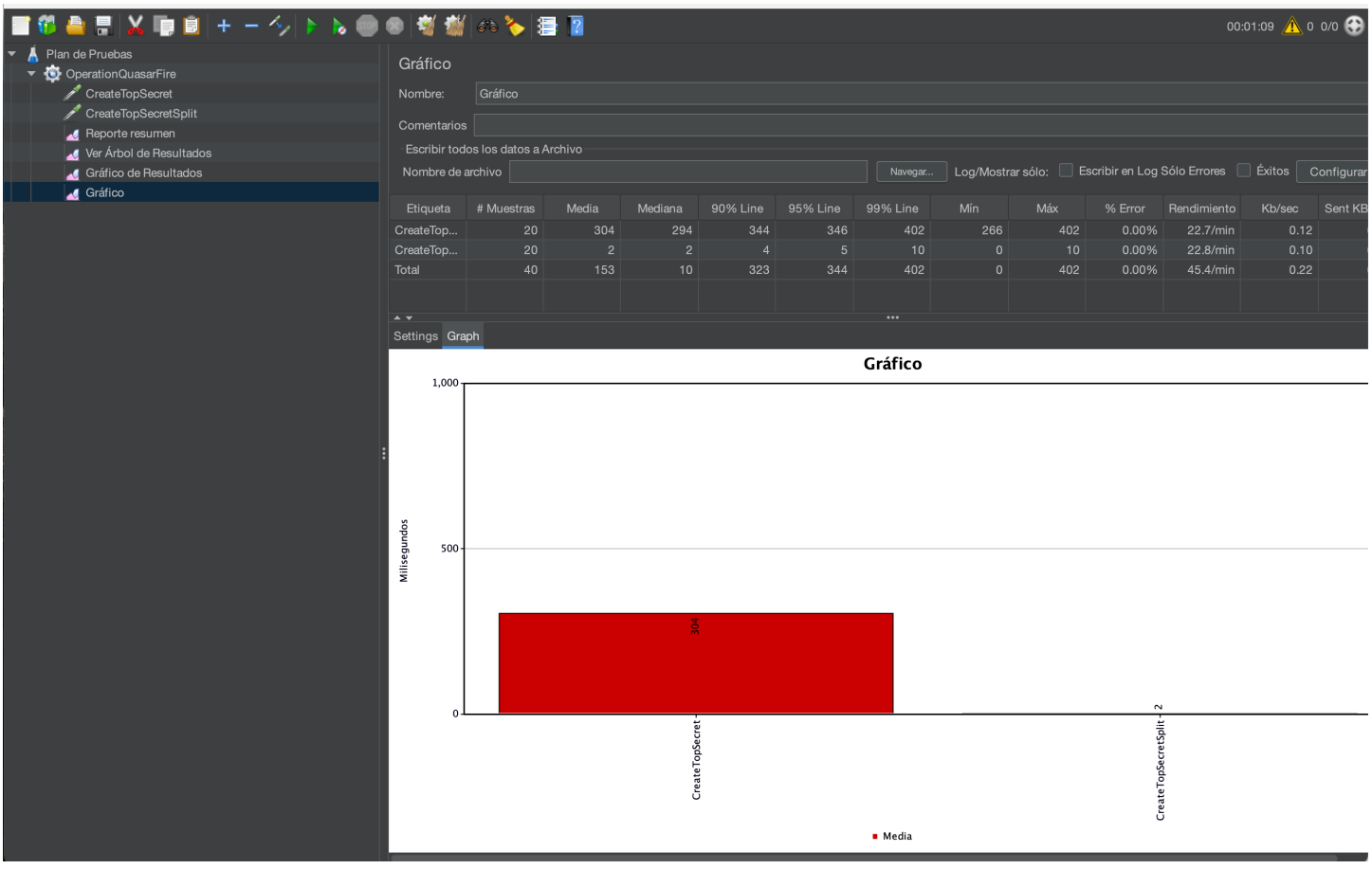
- Result


00:01:09 0 0/0
Plan de Pruebas
OperationQuasarFire
CreateTopSecret
CreateTopSecretSplit
Reporte resumen
Ver Árbol de Resultados
Gráfico de Resultados
Gráfico
Reporte resumen
Nombre: Reporte resumen
Comentarios
Escribir todos los datos a Archivo
Nombre de archivo
Navegar...
Log/Mostrar sólo:
Escribir en Log Sólo Errores
Éxitos
Configurar
Etiqueta # Muestras Media Mín Máx Desv. Estándar % Error Rendimiento Kb/sec Sent KB/sec Media de Byt...
CreateTopSe... 20 304 266 402 33.32 0.00% 22.7/min 0.12 0.35 328.0
CreateTopSe... 20 2 0 10 2.04 0.00% 22.8/min 0.10 0.10 263.0
Total 40 153 0 402 152.83 0.00% 45.4/min 0.22 0.45 295.5
¿Incluir el nombre del grupo en la etiqueta?
Guardar la tabla de datos
Guardar la cabecera de la tabla


Reporte resumen.jmx (/Users/juansebastiansanchez/Downloads/Reporte resumen.jmx) - Apache JMeter (5.6.3)
00:01:09 0 0/0
Plan de Pruebas
OperationQuasarFire
CreateTopSecret
CreateTopSecretSplit
Reporte resumen
Ver Árbol de Resultados
Gráfico de Resultados
Gráfico
Ver Árbol de Resultados
Nombre: Ver Árbol de Resultados
Comentarios
Escribir todos los datos a Archivo
Nombre de archivo
Navegar...
Log/Mostrar sólo:
Escribir en Log Sólo Errores
Éxitos
Configurar
Buscar:
Sensible a mayúsculas
Expresión regular
Search
Reset
JSON Path Tester
JSON Path Tester
{
"result": {
"details": [
{
"detail": "",
"message": "OK",
"internalCode": "200"
}
],
"source": "Find position to message"
},
"position": {
"x": 261.41962,
"y": 1105.2816,
"message": "este es un mensaje secreto"
}
}
JSON Path Expression
Test
Scroll automatically?


00:01:09 0 0/0
Plan de Pruebas
OperationQuasarFire
CreateTopSecret
CreateTopSecretSplit
Reporte resumen
Ver Árbol de Resultados
Gráfico de Resultados
Gráfico
Ver Árbol de Resultados
Nombre: Ver Árbol de Resultados
Comentarios
Escribir todos los datos a Archivo
Nombre de archivo
Navegar...
Log/Mostrar sólo:
Escribir en Log Sólo Errores
Éxitos
Configurar
Buscar:
Sensible a mayúsculas
Expresión regular
Search
Reset
JSON Path Tester
JSON Path Tester
{
"result": {
"details": [
{
"detail": "",
"message": "OK",
"internalCode": "200"
}
],
"source": "Message and distance added successfully"
}
}
JSON Path Expression
Test
Scroll automatically?





Repository explanation

Introduction

In this repository you will find the source code and documentation related to the solution to the "Operation Quasar Fire" challenge. This challenge involves creating a program in Golang that can determine the position and distress message emitted by a ship in distress, using information provided by three satellites.

- Repository Structure

This repository is structured following the principles of hexagonal architecture, ensuring a clear separation of the application, domain, and infrastructure layers. Each layer in the repository serves a specific purpose:

  - CMD Layer

The data access layer in a software system plays a crucial role in interacting with the underlying database. In our context, we are using a PostgreSQL database to store information related to satellites and their messages. NewPostgreSQLDB function, the NewPostgreSQLDB function is responsible for establishing a new connection to the PostgreSQL database and configuring it appropriately for use. Here is a detailed description of what this function does:

- Package Import: Start by importing the packages necessary to interact with PostgreSQL. We use the database/sql package to handle interaction with the database and the "github.com/lib/pq" package to register the PostgreSQL driver.
- Connection String Configuration: The function then takes connection configuration parameters such as host address, port, database name, username, and password. Use these parameters to generate a correctly formatted connection string.
- Establishing the connection: Use the sql.Open("postgres", connStr) function to open a new connection to the PostgreSQL database using the generated connection string.
- Connection verification: Once the connection is opened, db.Ping() is executed to verify if the connection to the database is successful. If the connection fails, an error message is printed and the connection is closed.
- Table creation: If the connection is successful, the function executes a SQL query to create the "Satellites" table in the database, if it does not already exist. This table has three columns: "name", "distance" and "message".
- Error Handling: At every step of the connection and configuration process, error checks are performed to ensure that any issues are handled properly. If an error occurs at any stage, an error message is printed and the connection is closed before returning the error.
- Connection Return: If the entire process completes successfully, the function returns a pointer to the database connection (*sql.DB), which is ready to be used by other parts of the program.

- Infrastructure Layer

  - Services and Routing Layer

In our Starship Communications Resolver system, the services and routing layer is responsible for handling HTTP requests, interacting with the business layer, and sending appropriate responses to clients.

  - Route Registration and Server Creation

The RegisterRoutes2 function is responsible for registering the API routes and their corresponding handlers to the Gin router. This function takes an object of type gin.Engine and a database connection as input parameters.

First, a cache system is configured with an expiration duration and a clearing interval. The repositories and services needed to handle the incoming requests are then initialized. One handler is created for the /topsecret/ endpoint and another for the /topsecret_split/:satellite_name endpoint. These handlers are bound to specific functions that process requests and return corresponding responses.

On the other hand, the CreateServer function is used to configure and create a Gin server. This function takes a database connection as a parameter and establishes a JWT middleware for authentication. It also applies CORS middleware to handle cross-origin requests. It then registers the routes using the RegisterRoutes2 function and returns the server configured and ready to run.

Finally, the RunServer function starts the created server on port 8080, allowing the application to start listening and processing incoming requests.

  - Endpoint Controllers

The topsecretHandler and topsecretSplitHandler handler handle incoming requests for the /topsecret/ and /topsecret_split/:satellite_name endpoints, respectively. These controllers are responsible for parsing the input data, invoking the corresponding use cases from the business layer, and sending the appropriate responses to the client.

Each controller checks the validity of the input data and calls the relevant use cases to process the request. If any errors occur during processing, an appropriate error response is returned to the client. Otherwise, a successful response is sent along with the requested data.

In summary, the services and routing layer provides the communication interface between the HTTP API and the underlying business logic, allowing our application to interact with clients efficiently and effectively.

- Authentication Middleware

In our Starship Communications Resolver system, the JWT middleware is responsible for validating and authenticating incoming requests using JWT tokens (JSON Web Token). This middleware ensures that requests are authorized and that only authenticated users can access protected API resources.

The JWTMiddleware structure defines the JWT middleware and contains a secretKey field that represents the secret key used to sign and verify JWT tokens. The NewJWTMiddleware function is used to create a new instance of the middleware with the secret key provided as a parameter.

The MiddlewareFunc function is the entry point of the middleware and is executed for each incoming request. It starts by retrieving the authorization header from the HTTP request. If the authorization header is absent, or the token format is invalid, an "Unauthorized" error is returned to the client.

If the authorization header is present and in the correct format (Bearer <token>), the middleware extracts the JWT token from the header and validates it. First, the header is split into two parts to extract the token. The jwt-go library is then used to parse and verify the token using the secret key provided when creating the middleware.

If the token is valid, the request is allowed to continue processing by calling c.Next(), which passes control to the next middleware or corresponding route handler. If the token is invalid, an "Unauthorized" error is returned to the client.

In short, JWT middleware ensures that requests are authenticated and authorized before accessing protected API resources. Provides an additional layer of security by validating JWT tokens and ensuring that only authorized users can access protected resources.

- Repository Layer

The CreateTaskTotal function is a critical part of the repository layer in the Starship Communications Resolver system. This layer is responsible for interacting with the database to perform data storage and retrieval operations. This document will explain in detail how this feature works and its role in the system.

The CreateTaskTotal function has the responsibility of inserting the satellite data into the database.

- pkg Layer

In the hexagonal architecture, the pkg (package) layer plays a fundamental role in providing an organized and modular structure for the system components. This layer is located in the center of the hexagon and contains the main business logic and domain rules of the system.

- Entity Layer

The Starship communications system uses several entities to represent satellite information and communication requests.

- Ports Layer

This package defines interfaces that act as ports to the outside of the hexagon in the hexagonal architecture. These interfaces establish contracts that must be fulfilled by particular implementations in other layers, such as external infrastructure or application logic.

- Response Layer

This package defines data structures used to format JSON responses on the system.

- Service Layer

These two blocks of code represent services that are part of the business layer of an application, specifically in the context of resolving spacecraft communications. Here is the explanation of each one:

- Service

This service implements the ports.CommunicationServices interface, which defines methods to obtain the location and message of a spacecraft.

The GetLocation method calculates the X and Y coordinates of the spacecraft location using a triangulation algorithm from the distances provided by three specific satellites (Kenobi, Skywalker, and Sato). These distances are used to calculate coordinates using geometric formulas.

The GetMessage method combines the messages received from all three satellites into one. It ensures that each word in the message is correctly added in the correct position using a "wrap" approach to a fixed length array.

- ServiceSplit

This service also implements the ports.CommunicationServices interface.

Like the main service, the GetLocation method calculates the X and Y coordinates of the spacecraft location using a similar triangulation algorithm.

The GetMessage method follows the same process as the parent service's counterpart method to combine messages received from satellites.

- Use Case Layer

These code blocks represent use cases within the hexagonal architecture of an application to solve spacecraft communications. Here is the explanation of each one:

- TopSecretUseCase

This use case is responsible for resolving the location and message of a spacecraft from the data provided by satellites.

The CreateTopSecret method takes a SatelliteRequest, which contains the information of the satellites that received the ship's signal.

It uses the repository to store data received from satellites.

It then traverses the satellite data to obtain the messages and respective distances.

It uses communication services to calculate the location and message of the spacecraft.

If the location or message cannot be determined, an error is returned. Otherwise, it returns a response containing the ship's position and message.

- TopSecretSplitUseCase

This use case handles the logic related to splitting the signal between multiple requests and collecting satellite data.

The CreateTopSecretSplit method is used to cache individual satellite data.

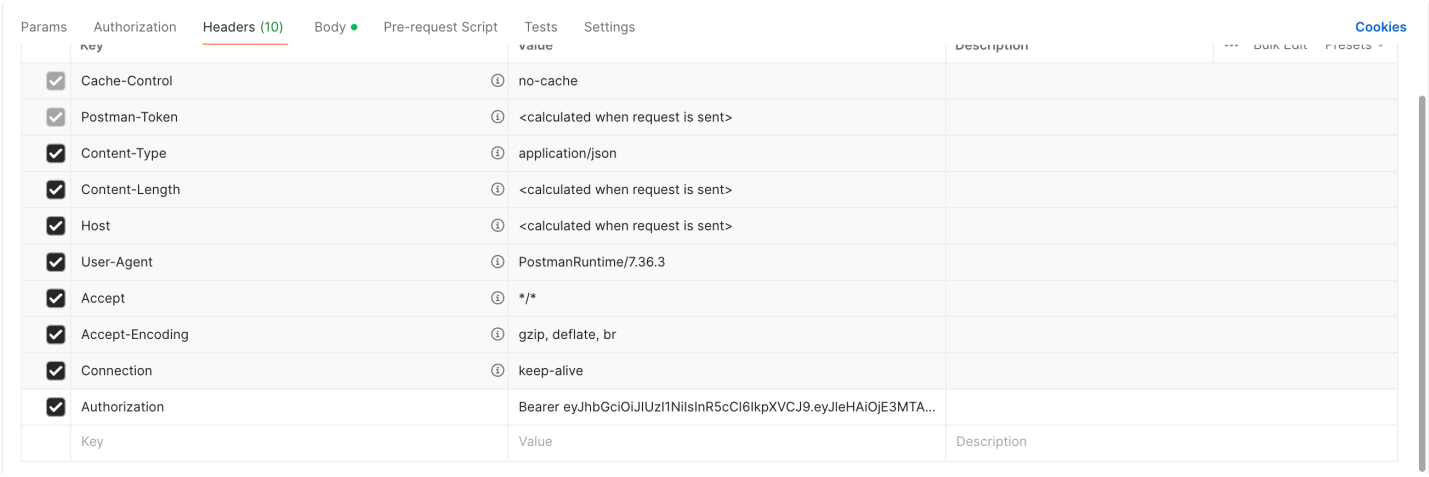
Checks if there are enough satellite positions in the cache to track the ship's location and message.

The GetAll method retrieves all satellite data from the cache, calculates the spacecraft's location and message, and returns a response.

It also provides a GetCachedData method to retrieve cached satellite data, and handles cases where the data is missing or invalid in the cache.

Example

- Header Configuration



Curl for generating a valid token

Curl: curl --location 'http://34.31.74.154:8080/auth-login/login' \

--header 'Authorization: ca3b18f4539ce3c81212d65d7c641009e8301c645967c9d28129f29d8d6514c5' \

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

--data-raw '{

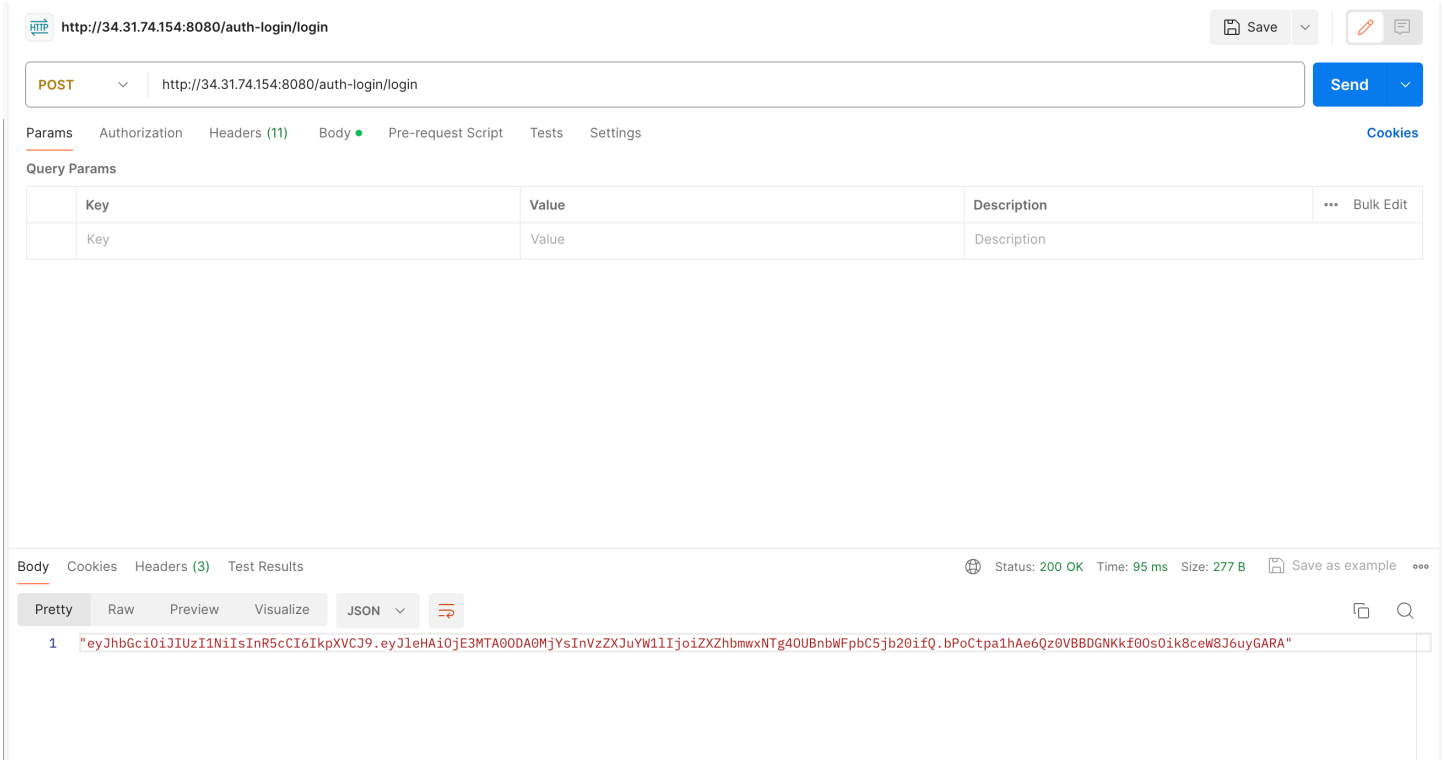
"username":"evanl15889@gmail.com",

"password":"$2a$10$/X11tnOBps3taWdz6gUx4.iMOo8gtiqYhTw1r0lBtmlPGHi7XL9XG"

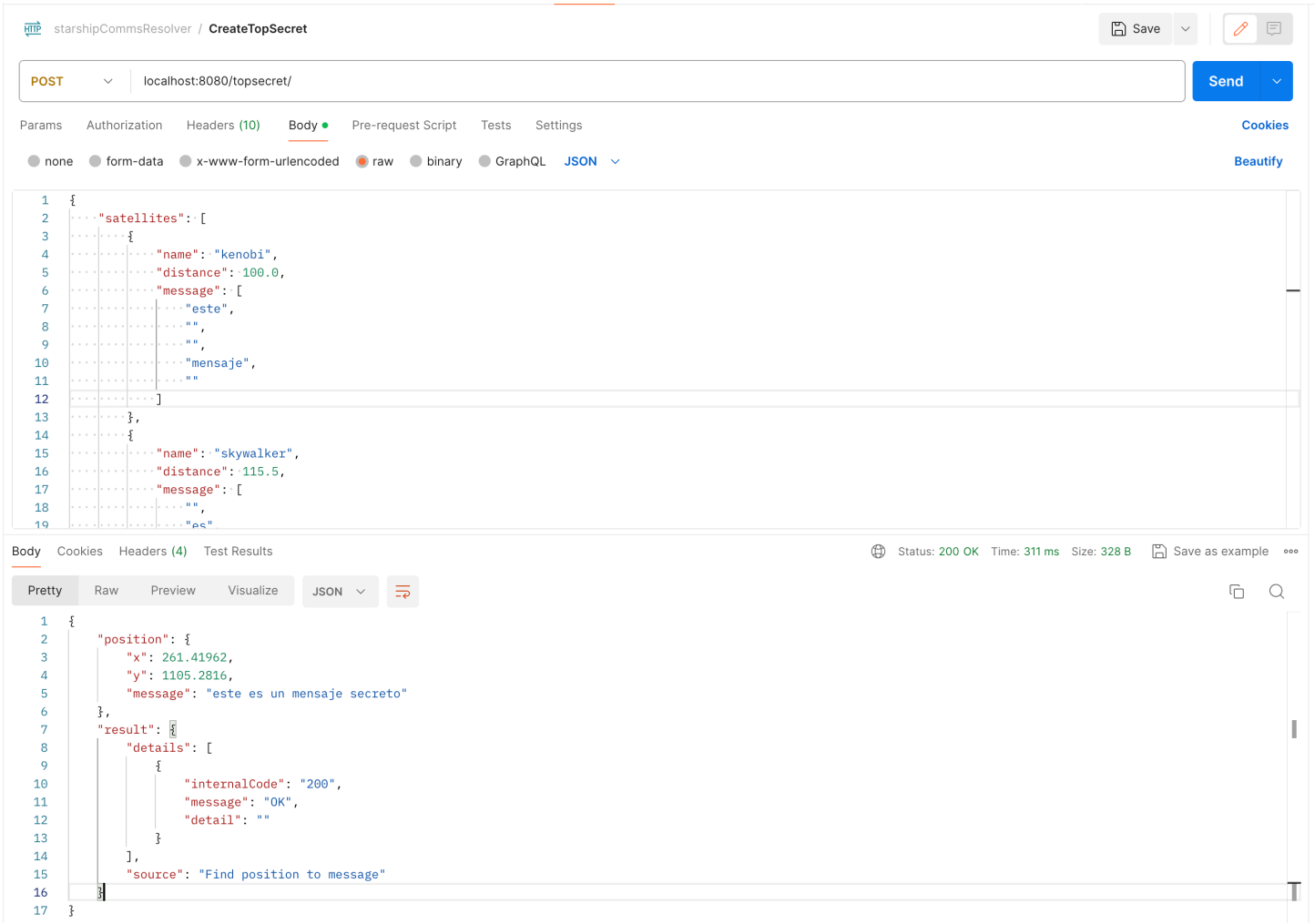
}

'

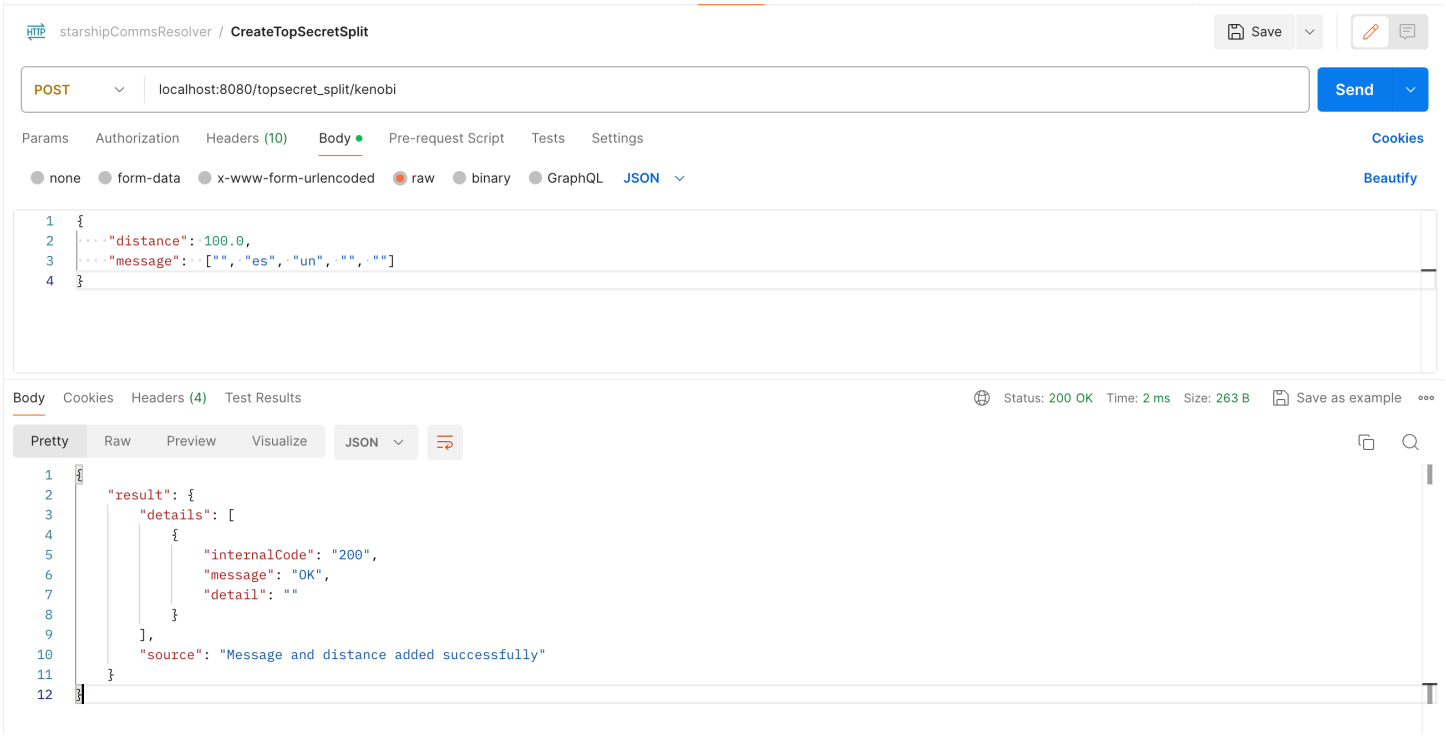
- Get Token



- CreateTopSecret



- CreateTopSecretSplit



- GetTopSecret