**DESARROLLO DE API REST**

para el levantamiento de los servicios se seleccionaron las siguientes tecnologías:

* **NodeJS/Typescript**: como lenguaje de lógica del servicio.
* **AWS**: como infraestructura en nube, nuestro proveedor de servicios, específicamente:
  + **AWS Lambda**: para almacenar y ejecutar la lógica del servicio.
  + **AWS APIGateway**: para exponer el servicio.
  + **AWS IAM**: para configurar permisos.
  + **AWS Dynamo**: para configurar la base de datos.
* **Jest**: como generador de pruebas unitarias y cobertura.
* **GitHub**: como controlador de versiones y repositorio.
* **Lint/Prettier**: como formateador de código.

**HISTORIAS DE USUARIO**

A continuación, haremos un levantamiento de historias épicas:

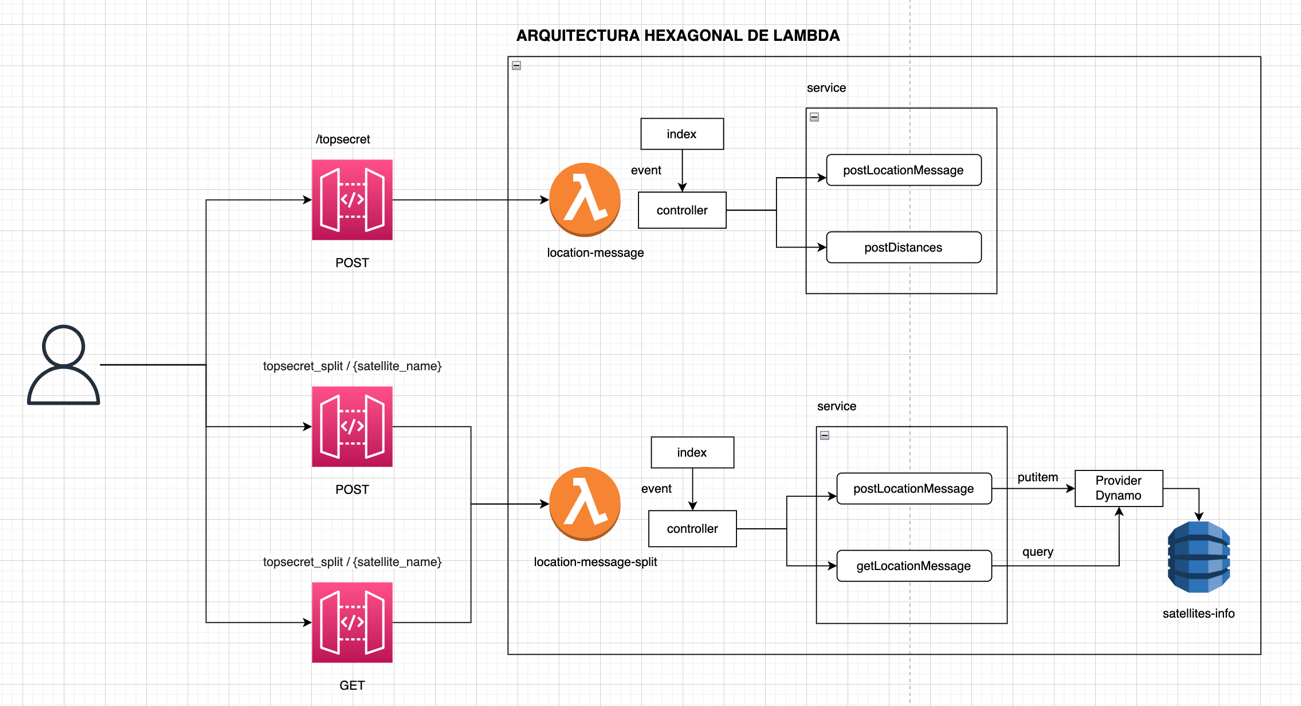
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **API REST PARA PROCESAR ARREGLO DE SATELITES** | | | | | |
| **NUMERO HU** | **ROL** | **CARACTERISTICA** | **RAZON** | **#** | **CRITERIO DE ACEPTACION** |
| ML-1 | Yo como Cliente | Necesito consumir un servicio POST que me permita enviar un arreglo de satélites | Para detectar si la posición y el mensaje enviado | 1 | El arreglo debe ser un arreglo de tamaño 3 |
| 2 | El arreglo debe contener los satellites de nombre: \* kenobi, skywalker y sato |
| 3 | Si se logra armar el mensaje y encontrar la posición, se retorna la información retornando un código 200 |
| 4 | Si no se logra determinar la posición se debe responder con código 404 |
| 5 | Cobertura mayor a 80% |
| 6 | Código fuente en repositorio GitHub |
| 7 | Instrucciones de uso en archivo README |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **API REST PARA ALMACENAR SATELITE** | | | | | |
| **NUMERO HU** | **ROL** | **CARACTERISTICA** | **RAZON** | **#** | **CRITERIO DE ACEPTACION** |
| ML-2 | Yo como Cliente | Necesito consumir 1 servicio POST que me permitan enviar por la url el nombre del satelite y además almacenar la información enviada de manera independiente a cada satélite | Almacenar la información de manera independiente | 1 | Almacenar correctamente el dato enviado |
| 2 | Cobertura mayor a 80% |
| 3 | Código fuente en repositorio GitHub |
| 4 | Instrucciones de uso en archivo README |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **API REST PARA PROCESAR INFORMACION ALMACENADA** | | | | | |
| **NUMERO HU** | **ROL** | **CARACTERISTICA** | **RAZON** | **#** | **CRITERIO DE ACEPTACION** |
| ML-3 | Yo como Cliente | Necesito consumir 1 servicio GET que me permita en la. url enviar el nombre del satelite que deseo consumir | Obtener la información de la. Posición y el mensaje si es posible | 1 | Calcular la posición y obtener el mensaje |
| 2 | Cobertura mayor a 80% |
| 3 | Código fuente en repositorio GitHub |
| 4 | Instrucciones de uso en archivo README |

**ARQUITECTURA**

Para la construcción de la solución se planteo la siguiente arquitectura:

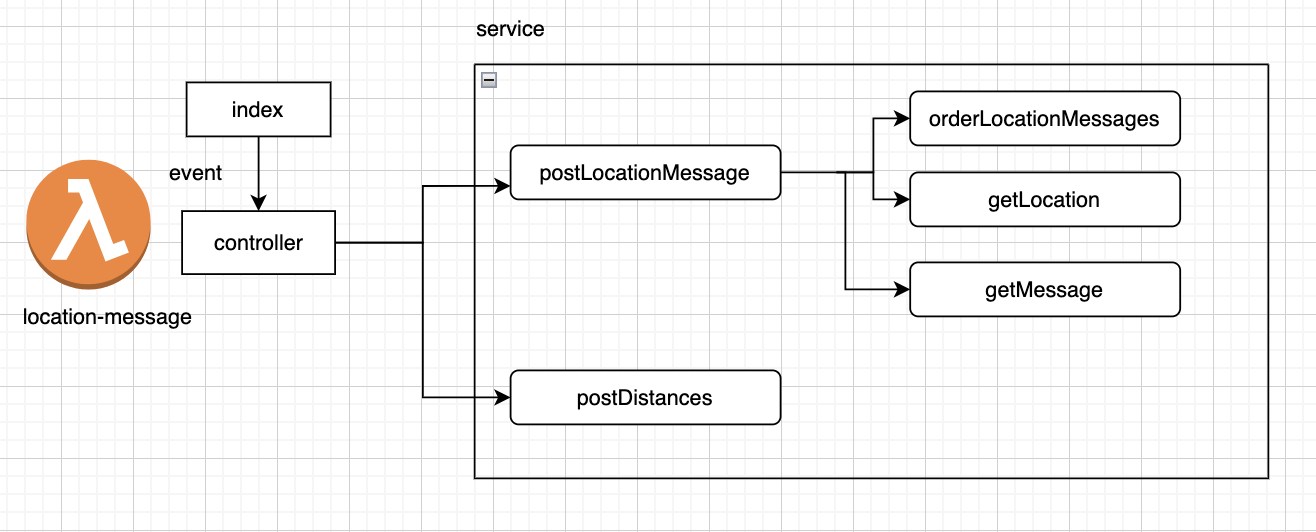
****

Como podemos ver se planteo el esquema de micro servicios en el cual cada lambda se encarga únicamente de un solo proceso, de esta manera se decidió romper un sistema monolítico y separar los servicios en lambdas independientes.

El código de cada lambda esta estructurado bajo el esquema de una arquitectura hexagonal, en la cual se separan los métodos de su implementación (patrón de diseño Facade), de esta manera podemos asegurar un código escalable y flexible a cambios de implementación. También teniendo en cuenta los principios de Clean Code, por lo cual el código se hace altamente legible y fácil de comprender.

**ALGORITMOS**

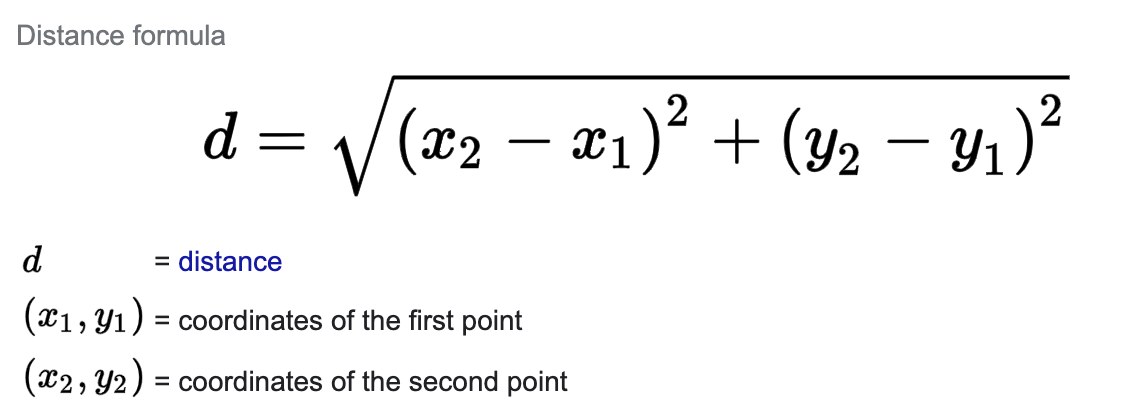
Para el algoritmo que pudiera calcular la posición y el mensaje se planteó la siguiente solución:



* El servicio **postLocationMessgae** es llamado por el controlador y este realiza las siguientes operaciones:

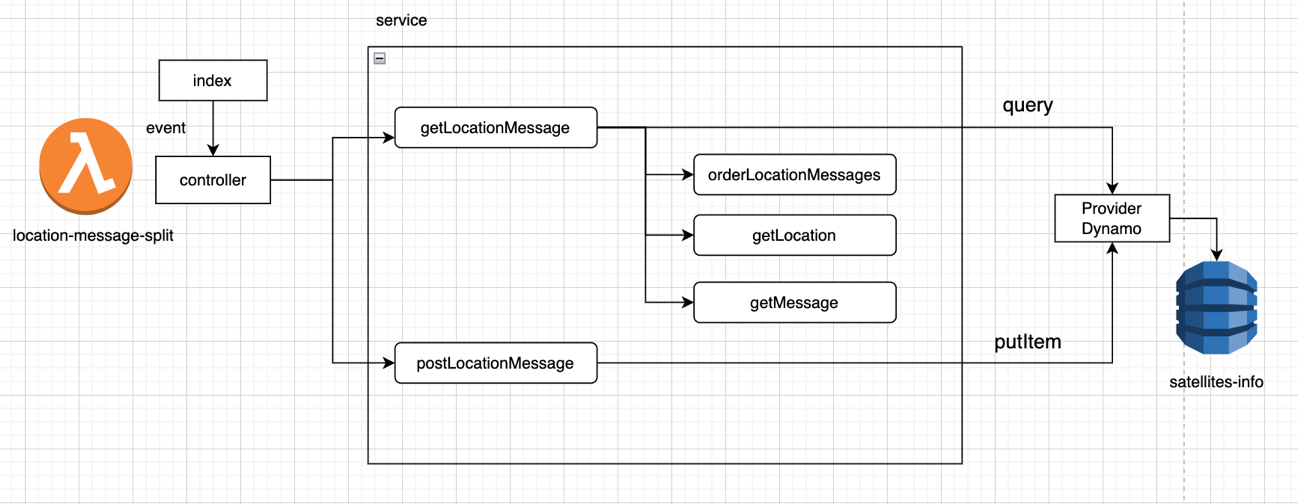
1. Valida que los datos de entrada que sean correctos.
2. Pasa los datos de los satélites por la función **orderLocationMessages**, el cual se encarga de armar un arreglo de posiciones y otro arreglo de mensajes.
3. Enviamos los arreglos obtenidos en el item 2, a las funciones **getLocation** y **getMessage**, los cuales son los encargados de hacer el cálculo de la posición y el armado del mensaje final respectivamente.
4. Retornamos la información con la estructura solicitada.

* La función **getLocation** contiene una serie de fórmulas matemáticas que permiten el cálculo de la posición tomando como base la fórmula de la distancia entre dos puntos:



1. De esta manera, realizamos los cálculos con las distancias ingresadas, calculamos en primera instancia por medio de un sistema de ecuaciones el valor de y1 y finalmente calculamos el valor de x1.
2. Por ultimo evaluamos ambos valores en las ecuaciones y el resultado debe ser el mismo en todas, de esta manera validamos que la posición (x1,y1) obtenidas son correctas, de no serlo el sistema no tiene solución y por tanto la posición no puede ser determinada.

* La función **getMessages** básicamente recorre las posiciones de cada uno de los arreglos y dependiendo cual contenga una palabra valida, esta se agrega al arreglo final que conformara el mensaje, una de las condiciones es que la palabra valida no sea igual a la última que ya contiene el arreglo, de esta manera nos aseguramos que no se repitan palabras que esten repetidas en más de un arreglo de mensaje.
* La función **postDistances** no forma parte de la solucion como tal, la configure con el fin de obtener los valores de las distancias de cada uno de los satélites al ingresar una posición especifica, de esta manera los valores obtenidos podemos ingresarlos en el ejercicio real y corroborar el resultado de las coordenadas obtenidas.

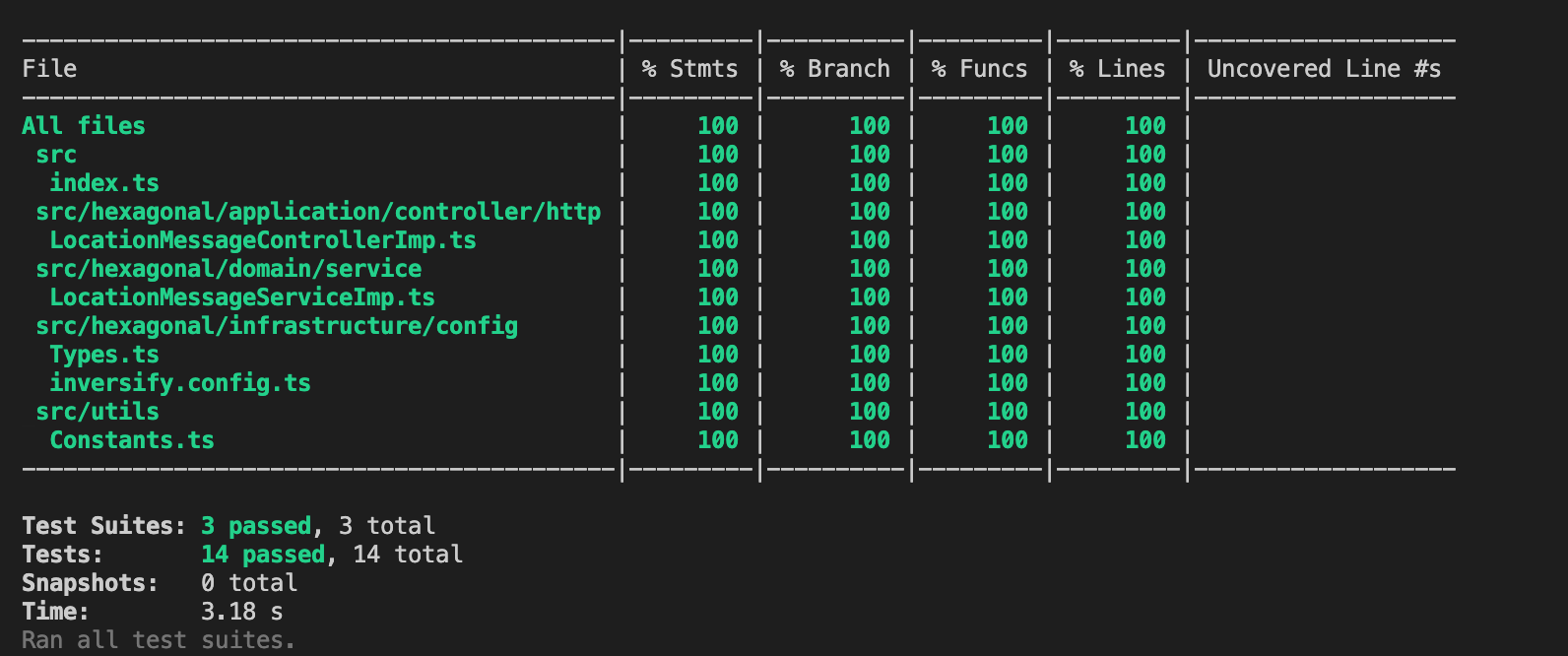


* El servicio **postLocationMessage** se conecta a un proveedor de Dynamo DB y por medio del método **putItem** almacena en la tabla **satellites-info** la información enviada.
* El servicio **getLocationMessage** se conecta a un proveedor de Dynamo DB y por medio del método **query** obtienes los últimos datos ingresados a cada satélite, y de esta manera realiza el cálculo de la posición y el mensaje con los métodos anteriormente mencionados.

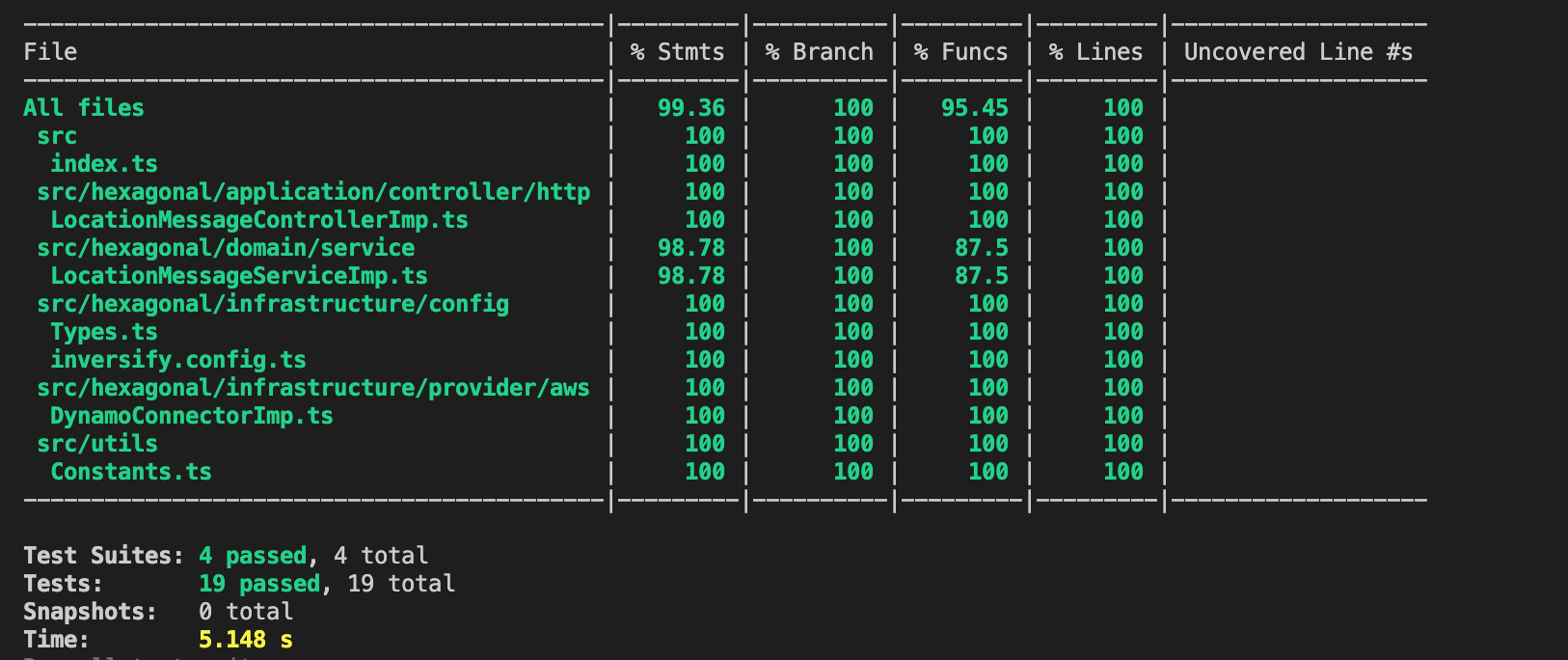
**TEST Y COBERTURA**

Se configuraron y ejecutaron las pruebas unitarias del código arrojando los siguientes resultados:

* Código de la Lambda location-message



* Código de la lambda location-message-split



**RESPUESTAS DE LAS API**

|  |  |  |
| --- | --- | --- |
| **RESPUESTA DE LA API LOCATION-MESSAGE POST** | | |
| **codigo** | **descripcion** | **respuesta** |
| 200 | Respuesta cuando la posición y el mensaje logran ser determinados | {  "position": [  {  "x": -500,  "y": 99.7  }  ],  "message": "este es un mensaje secreto para la gente de bien"  } |
| 404 | Respuesta cuando la posición no puede ser determinada | {  "position": "No fue posible determinar la posicion con las distancias ingresadas",  "message": "este es un mensaje secreto para la gente de bien"  } |

|  |  |  |
| --- | --- | --- |
| **RESPUESTA DE LA API LOCATION-MESSAGE-SPLIT POST** | | |
| **codigo** | **descripcion** | **respuesta** |
| 200 | Respuesta exitosa | {  "message": "Informacion almacenada con exito",  "informacion": {  "distance": 1000,  "message": ["Informacion"]  }  } |
| 404 | Respuesta ante algún error | "El nombre de satellite enviado no es valido" |

|  |  |  |
| --- | --- | --- |
| **RESPUESTA DE LA API LOCATION-MESSAGE-SPLIT GET** | | |
| **codigo** | **descripcion** | **respuesta** |
| 200 | Respuesta exitosa | {  "position": [  {  "x": -500,  "y": 99.7  }  ],  "message": "este es un mensaje secreto para la gente de bien"  } |
| 404 | Respuesta cuando no se han suministrado los datos de los3 satélites, suficientes para el calculo | "Aun no es posible determinar la posicion ni el mensaje hasta no suministrar los datos de los 3 satelites" |
| 404 | Respuesta si el nombre de satélite enviado es invalido | "El nombre de satellite enviado no es valido" |