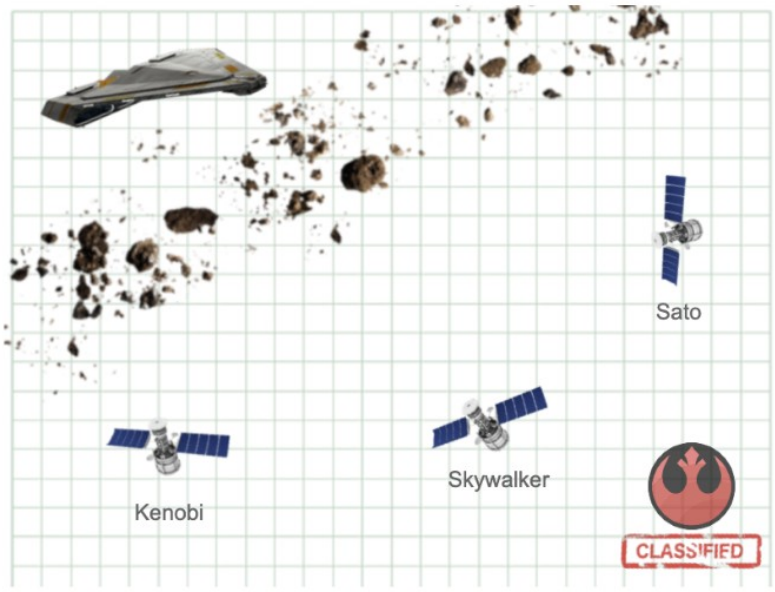
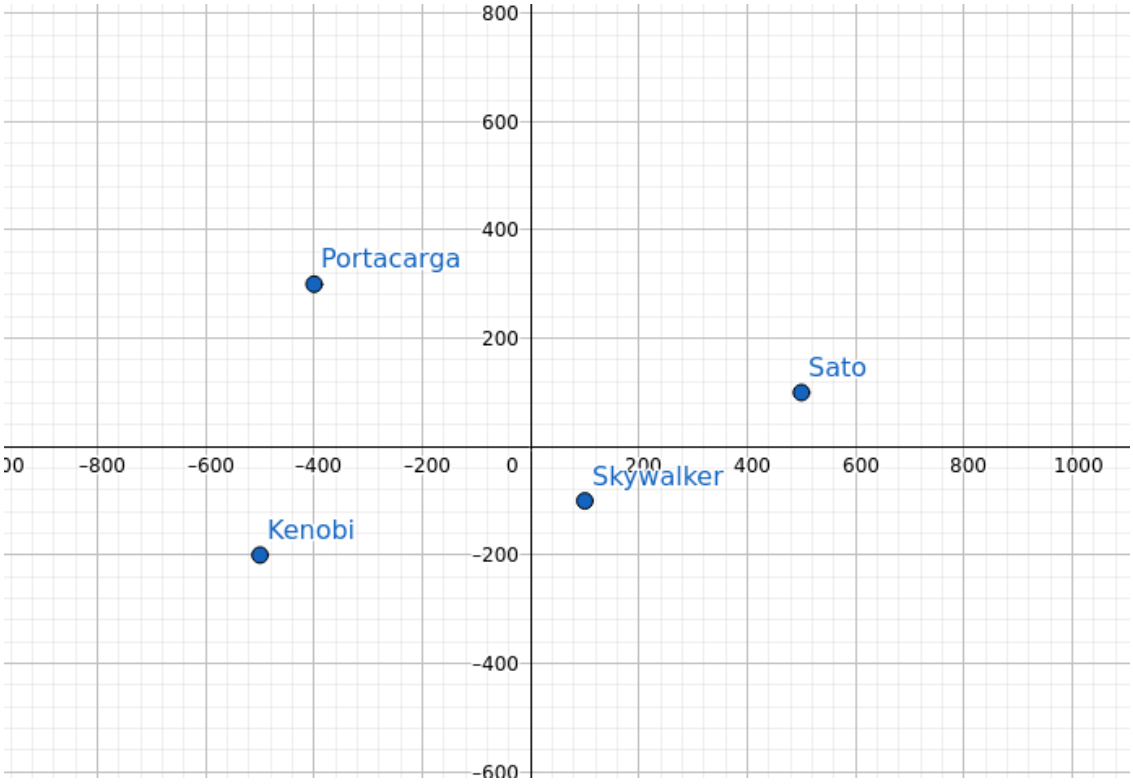
Documentación Challenge Quasar, para el ingreso de C Emmanuel Moreno Torres a Mercado Libre.

El sistema debe calcular la posición y el mensaje secreto de la nave portacarga imperial que está emitiendo un llamado de auxilio a 3 satélites de la Alianza Rebelde.



La imagen anterior se puede representar en plano cartesiano de la siguiente forma:

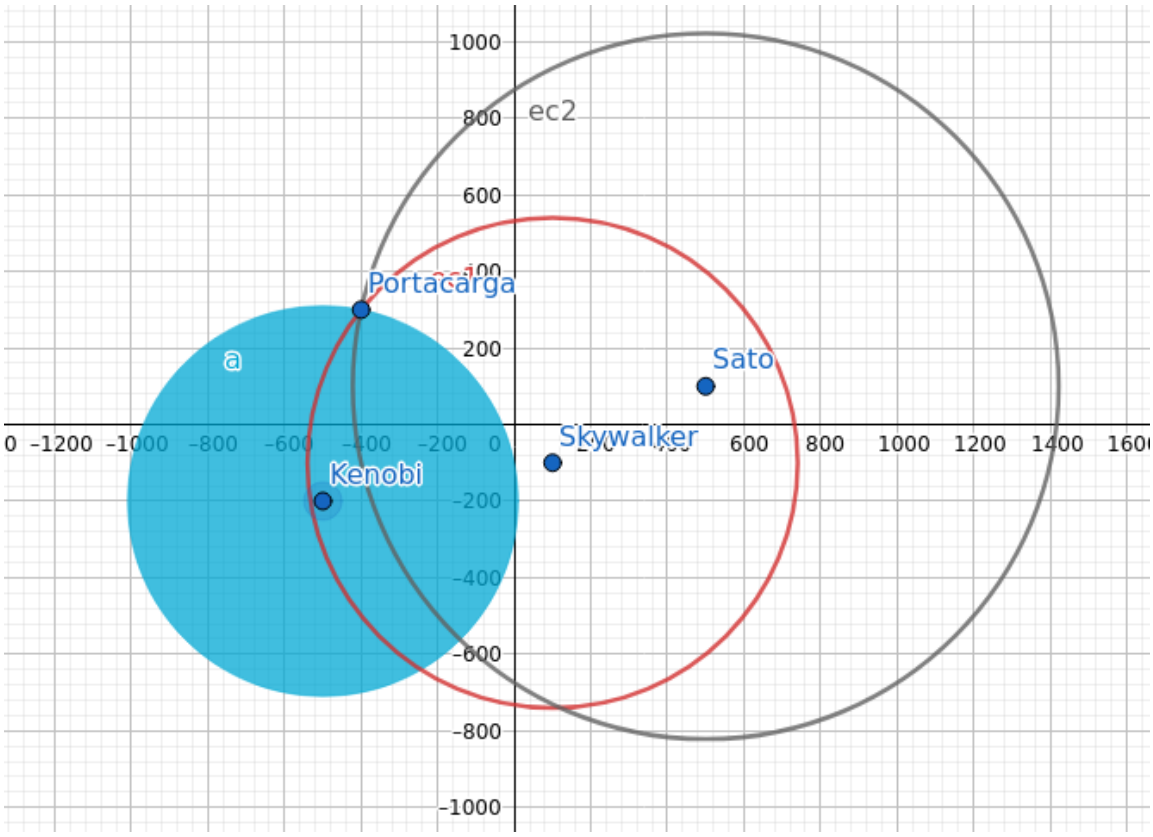


La información disponible que se tiene para dicho cálculo es la distancia y el mensaje incompleto del portacarga imperial hacia cada satélite, representado de la siguiente manera:


```
{
   "name":"kenobi",
   "distance":509.9019513592785,
   "message":["este","","","mensaje",""]
}
```


Donde el campo “**name**” representa el satélite a donde se está enviando la información, “**distance**” es la distancia a la que se encuentra la nave portacarga imperial del satélite, y “**message**” un arreglo de cadenas con el mensaje incompleto, éste debe de tener la misma longitud en cada uno de los 3 llamados.

Para realizar el cálculo de las coordenadas del portacarga imperial, se decidió graficar una circunferencia al rededor de cada satélite, usando como radio la distancia al portacarga imperial, así, el punto de intersección de los 3 círculos, es la posición del portacarga imperial.



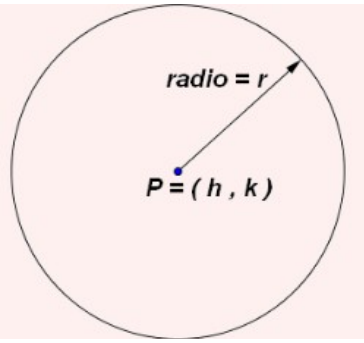
En éste ejemplo, usaremos los valores:

name: "kenobi"
distance: 509.9019513592785
message: ["este","","","mensaje",""]

name: "skywalker"
distance: 640.3124237432849
message: ["","es","","","secreto"]

name: "sato"
distance: 921.9544457292888
message: ["este","","un","",""]

Para comenzar, se debe obtener la ecuación general de la circunferencia de cada satélite.



Sea la circunferencia de radio r
y centro P(h , k)
La Ecuación General es:

$$x^2 + y^2 + Ax + By + C = 0$$

Donde:

$$A = -2h$$
$$B = -2k$$
$$C = h^2 + k^2 - r^2$$

Ésto se hace con un método genérico (X^2 y Y^2 al siempre ser 1, se omitirán en el mapeo de los pojos en todo el sistema).

```java
public Equation getCircumferenceEquation(Coordinates coordinates, double radius) {

    double a = coordinates.getX() * -2;
    double b = coordinates.getY() * -2;
    double c = Math.pow(coordinates.getX(), 2) + Math.pow(coordinates.getY(), 2) - Math.pow(radius, 2);

    return new Equation(Util.roundingDecimals(a,2),Util.roundingDecimals(b,2),Util.roundingDecimals(c,2),0);
}
```

Una vez teniendo las ecuaciones de la circunferencia de los 3 satélites, se calculan las intersecciones entre ellos, que en éste caso son Kenobi → Skywalker, Skywalker → Sato y Kenobi → Sato.

Para éste ejemplo, la ecuación de la circunferencia Kenobi es la siguiente:

$$x^2 + y^2 + 1000x + 400y + 30000$$

Y la de Skywalker es la siguiente:

$$x^2 + y^2 - 200x + 200y - 390000$$

Para la intersección entre dos circunferencias, primero se suman las ecuaciones generales de la circunferencia, ésto se hace multiplicando la segunda ecuación (-1) y efectuando la suma.

$$\begin{array}{r} x^2 + y^2 + 1000x + 400y + 30000 \\ -x^2 - y^2 + 200x - 200y + 390000 \\ \hline 1200x + 200y + 420000 \end{array}$$

Luego se despeja x:

$$1200x = -200y - 420000$$
$$x = -0.16666y - 350$$

Dentro del método genérico el código hasta éste punto es el siguiente:

```
//The equations are added
//The second equation is converted to negative
Equation circleEquationTwo = new Equation();
circleEquationTwo.setCoefficientX(circleEquationTwoNormal.getCoefficientX() * -1);
circleEquationTwo.setCoefficientY(circleEquationTwoNormal.getCoefficientY() * -1);
circleEquationTwo.setIndependentTermC(circleEquationTwoNormal.getIndependentTermC() * -1);

//TODO validar x y y
Equation lineEquation = new Equation();
lineEquation.setCoefficientX(circleEquationOne.getCoefficientX() + circleEquationTwo.getCoefficientX());
lineEquation.setCoefficientY(circleEquationOne.getCoefficientY() + circleEquationTwo.getCoefficientY());
lineEquation.setIndependentTermC(circleEquationOne.getIndependentTermC() + circleEquationTwo.getIndependentTermC());

//Solve for X
lineEquation.setCoefficientY((lineEquation.getCoefficientY() / lineEquation.getCoefficientX()) * -1);
lineEquation.setIndependentTermC((lineEquation.getIndependentTermC() / lineEquation.getCoefficientX()) * -1);
lineEquation.setCoefficientX(lineEquation.getCoefficientX() / lineEquation.getCoefficientX());
```

Ahora la variable x se sustituye en la primera ecuación de la circunferencia que obtuvimos, en éste caso la del satélite “kenobi”

$$(- 0.16666y - 350)^2 + y^2 + 1000(- 0.16666y - 350) + 400y + 30000$$

Se resuelve el término cuadrático $(- 0.16666y - 350)^2$ mediante la formula:

$$(a + b)^2 = a^2 + 2ab + b^2$$

Por lo tanto:

$$(- 0.16666y - 350)^2 = 0.0277y^2 + 116.662y + 122500$$

Y se resuelve la segunda operación $1000(- 0.16666y – 350)$ multiplicando los términos:

$$1000(- 0.16666y – 350) = -166.66y – 350000$$

Con lo cual quedaría la ecuación completa de la siguiente manera:

$$0.0277y^2 + 116.662y + 122500 + y^2 - 166.66y – 350000 + 400y + 30000$$

Y ya resuelta y simplificada:

$$1.0277y^2 + 350.002y – 197500 \rightarrow y^2 + 340.57y – 192176.71$$

Dentro del método genérico el código hasta éste punto es el siguiente:

```
//It is substituted into the general equation of the circumference
//The quadratic term is solved
ySquared = Math.pow(lineEquation.getCoefficientY(), 2);
independentTerm = Math.pow(lineEquation.getIndependentTermC(), 2);
y = 2 * lineEquation.getCoefficientY() * lineEquation.getIndependentTermC();

//The second operation is solved
yTwo = circleEquationOne.getCoefficientX() * lineEquation.getCoefficientY();
independentTermTwo = circleEquationOne.getCoefficientX() * lineEquation.getIndependentTermC();

//They add up
ySquared = ySquared + 1;
y = y + circleEquationOne.getCoefficientY() + yTwo;
independentTerm = independentTerm + independentTermTwo + circleEquationOne.getIndependentTermC();

//It simplifies
y = y / ySquared;
independentTerm = independentTerm / ySquared;
ySquared = ySquared / ySquared;
```

Dada una ecuación de segundo grado:

$$y^2 + 340.57y - 192176.71$$

Se procede a resolver por medio de la fórmula general.

$$\frac{-b \pm \sqrt{(b^2 - 4ac)}}{2a}$$

Se obtiene el discriminante:

$$(340.57)^2 - 4\,(1)\,(-192176.71) = 891606.1461$$

Se continua con el desarrollo de la fórmula:

$$\frac{-340.57 \pm \sqrt{(891606.1461)}}{2}$$

Se resuelven las 2 incógnitas, obteniendo los dos puntos de intersección entre las circunferencias:

x1 = - 400 x2 = - 243
y1 = 300 y2 = - 640

Dentro del método genérico el código hasta éste punto es el siguiente:

```
//Solve the quadratic equation
discriminating = Math.pow(y, 2) - (4 * ySquared * independentTerm);

//if(discriminating > 0) has two real roots
yFinalOne = ((-1 * y) + Math.sqrt(discriminating)) / (2 * ySquared);
yFinalTwo = ((-1 * y) - Math.sqrt(discriminating)) / (2 * ySquared);

//The intersection points are constructed
xFinalOne = (lineEquation.getCoefficientY() * yFinalOne) + lineEquation.getIndependentTermC();
xFinalTwo = (lineEquation.getCoefficientY() * yFinalTwo) + lineEquation.getIndependentTermC();

coordinatesList.add(new Coordinates(xFinalOne, yFinalOne));
coordinatesList.add(new Coordinates(xFinalTwo, yFinalTwo));
return coordinatesList;
```

Al tener los puntos de intersección, éstos se agregan a una lista, y ya teniendo los cálculos de los las intersecciones de las 3 circunferencias, se procede a compararlos, y el punto que más se repita, es el punto donde se encuentra la nave portacarga imperial.

```java
public Coordinates getCoordinates(List<Coordinates> coordinatesList) {

    LinkedList<String> coordinatesListS = new LinkedList<>();
    coordinatesList.stream().forEach(coordinates -> coordinatesListS.add(coordinates.getConcatString()));

    List<String> duplicateList = coordinatesListS.stream()
                .collect(Collectors.groupingBy(s -> s)).entrySet().stream()
                .filter(e -> e.getValue().size() > 1)
                .map(e -> e.getKey())
                .collect(Collectors.toList());

    String[] duplicatedArray = duplicateList.get(0).split(",");

    double x = Double.parseDouble(duplicatedArray[0]);
    double y = Double.parseDouble(duplicatedArray[1]);

    return new Coordinates(x, y);
}
```

Diagrama ER

El sistema cuenta con seguridad JWT implementada con Spring security, para lo cual se designaron 2 roles: “ADMIN” y ”USER”, con el cual el administrador puede crear otros usuarios de tipo USER para que puedan consumir los endpoints del sistema.

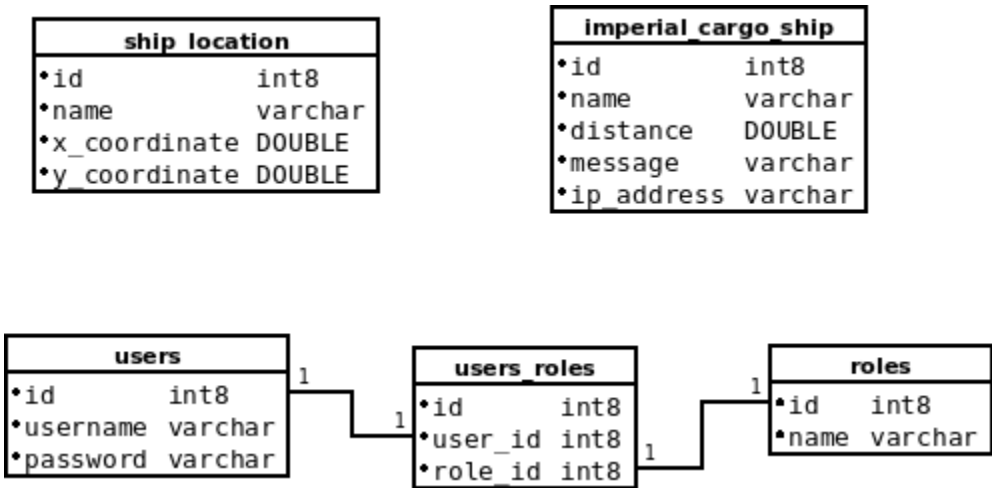


Tabla: ship_location
Descripción: Contiene las coordenadas y los satelites principales del sistema.

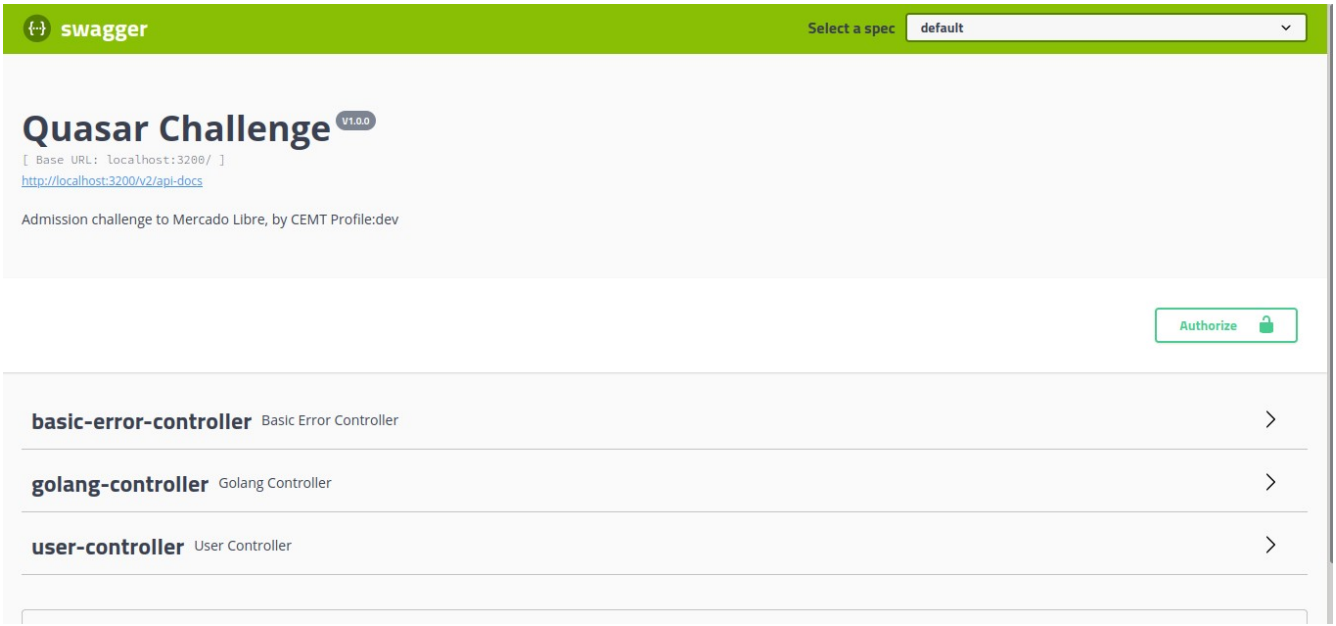
Clave	Nombre	Campo	Tipo	Descripción
PK	Identificador	id	int8	Número identificador del registro
	Nombre	name	varchar	Nombre del satélite al que va dirigido
	Distancia	distance	DOUBLE	Distancia del satélite
	Mensaje incompleto	message	varchar	Arreglo de cadenas, del mensaje incompleto
	Dirección IP	ip_address	varchar	Dirección IP del cliente

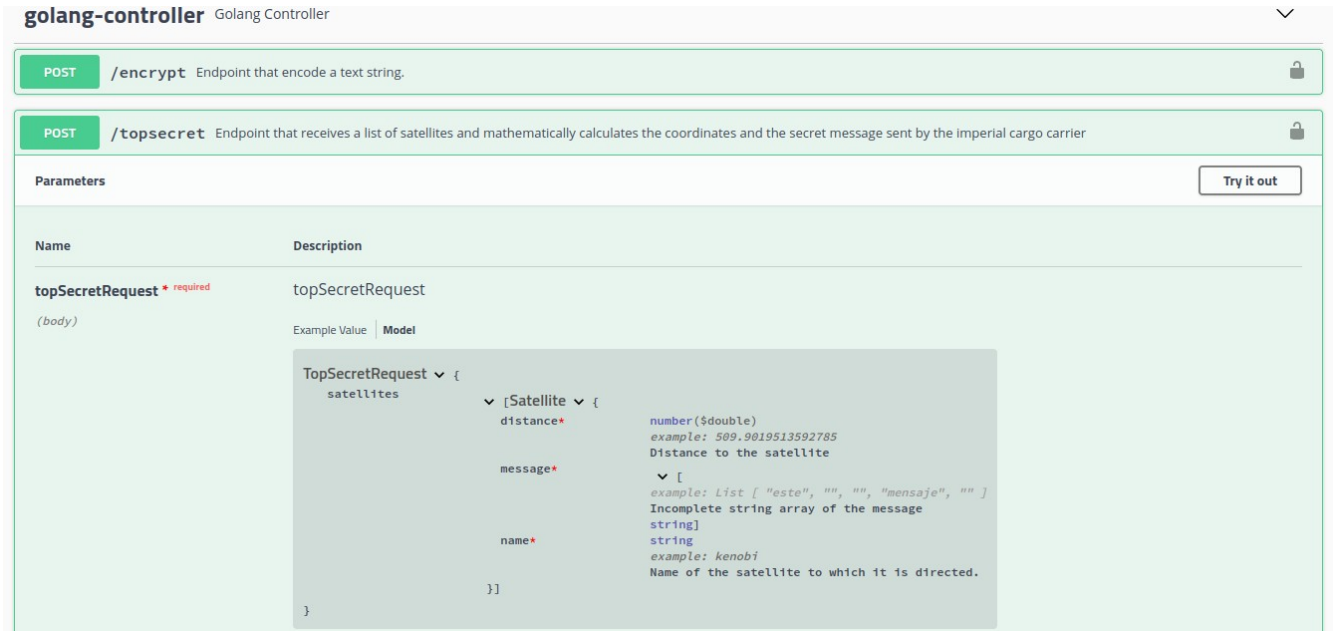
Tabla: imperial_cargo_ship
Descripción: Contiene la información del portacarga por cada satélite, para hacer los llamados independientes.

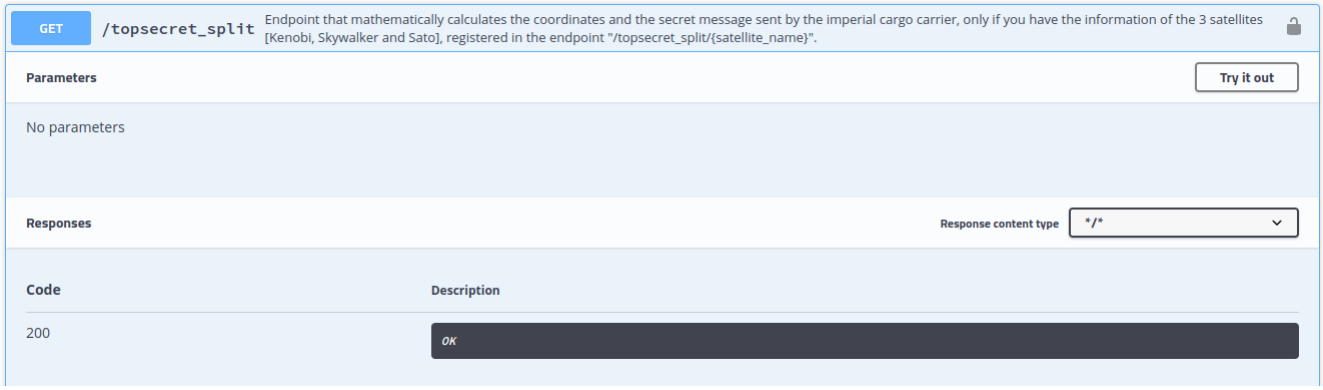
Clave	Nombre	Campo	Tipo	Descripción
PK	Identificador	id	int8	Número identificador del registro
	Nombre	name	varchar	Nombre del satélite
	Coordenada en x	x_coordinate	DOUBLE	Coordenada en x de la nave
	Coordenada en y	y_coordinate	DOUBLE	Coordenada en y de la nave

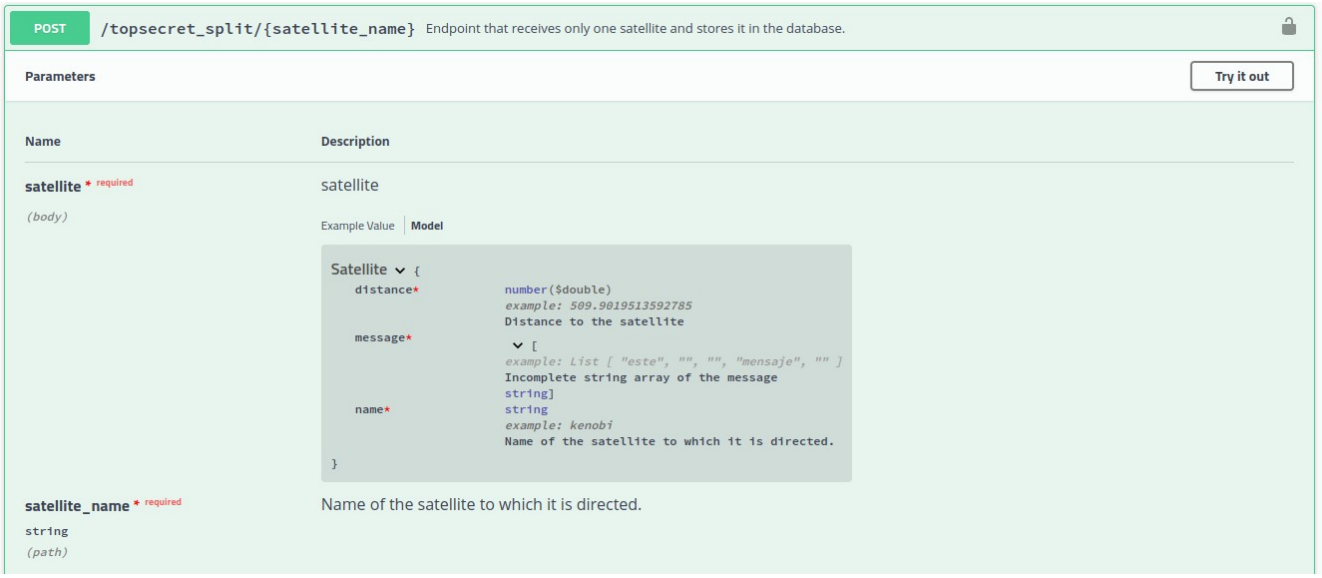
Swagger2

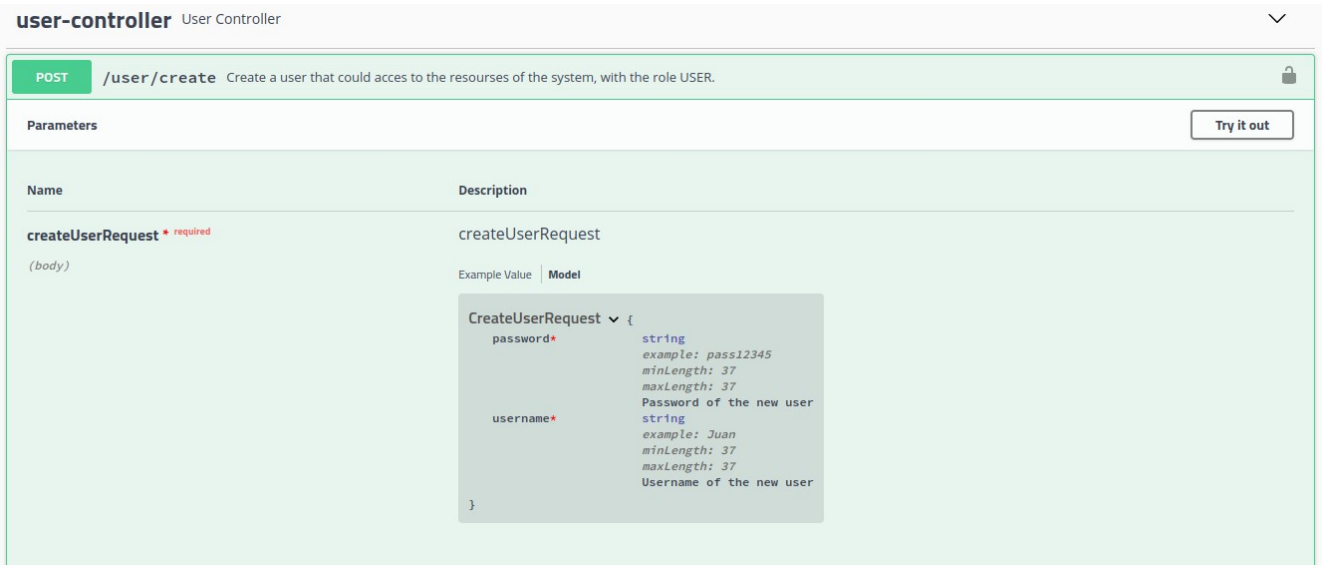
La documentación de los endpoints se realizó con la librería Swagger2






GET
/topsecret_split
Endpoint that mathematically calculates the coordinates and the secret message sent by the imperial cargo carrier, only if you have the information of the 3 satellites [Kenobi, Skywalker and Sato], registered in the endpoint "/topsecret_split/{satellite_name}".
Parameters
Try it out
No parameters
Responses
Response content type
/
Code
Description
200
OK


POST
/topsecret_split/{satellite_name}
Endpoint that receives only one satellite and stores it in the database.
Parameters
Try it out
Name
Description
satellite * required
(body)
satellite
Example Value
Model
Satellite {
distance* number($double)
example: 509.9019513592785
Distance to the satellite
message* [
example: List ["este", "", "", "mensaje", ""]
Incomplete string array of the message
string]
name* string
example: kenobi
Name of the satellite to which it is directed.
}
satellite_name * required
string
(path)
Name of the satellite to which it is directed.


user-controller
User Controller
POST
/user/create
Create a user that could acces to the resourses of the system, with the role USER.
Parameters
Try it out
Name
Description
createUserRequest * required
(body)
createUserRequest
Example Value
Model
CreateUserRequest {
password* string
example: pass12345
minLength: 37
maxLength: 37
Password of the new user
username* string
example: Juan
minLength: 37
maxLength: 37
Username of the new user
}

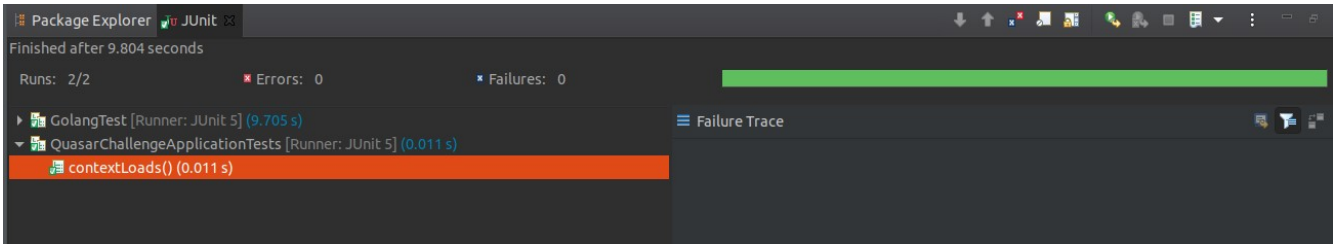
Testing

La ejecución de pruebas unitarias se realizó con Junit y Mockito, de la siguiente manera:

```java
@ExtendWith(MockitoExtension.class)
@AutoConfigureTestDatabase(replace=Replace.NONE)
public class GolangTest {

    @Mock(answer = Answers.CALLS_REAL_METHODS)
    private GolangService golangService;

    @Mock(answer = Answers.CALLS_REAL_METHODS)
    private Validator validator;

    @Mock
    private ShipLocationRepository shipLocationRepository;

    @Mock(answer = Answers.CALLS_REAL_METHODS)
    private MathOperations mathOperations;

    @BeforeEach
    void setUp() throws Exception {
        golangService.setTestsObjects(validator, shipLocationRepository, mathOperations);
    }
```

```java
    @Test
    public void topSecret() {

        when(shipLocationRepository.findByName(Constants.SHIP_KENOBI)).thenReturn(Optional.of(new ShipLocation((long) 1, "kenobi", -500, -200)));
        when(shipLocationRepository.findByName(Constants.SHIP_SKYWALKER)).thenReturn(Optional.of(new ShipLocation((long) 2, "skywalker", 100, -100)));
        when(shipLocationRepository.findByName(Constants.SHIP_SATO)).thenReturn(Optional.of(new ShipLocation((long) 3, "sato", 500, 100)));

        TopSecretRequest topSecretRequest = new TopSecretRequest();
        List<Satellite> satellites = new LinkedList<>();
        satellites.add(new Satellite("kenobi", 509.9019513592785, Arrays.asList("este","","","mensaje","")));
        satellites.add(new Satellite("skywalker", 640.3124237432849, Arrays.asList("","es","","","secreto")));
        satellites.add(new Satellite("sato", 921.9544457292888, Arrays.asList("este","","un","","")));
        topSecretRequest.setSatellites(satellites);

        ResponseEntity<?> response = golangService.topSecret(topSecretRequest);

        System.out.println(response.toString());
    }
```

Y éstas corrieron de forma satisfactoria:




```
11:56:45.804 [main] INFO com.quasar.service.GolangService - Total execution time topSecret 7
<200 OK OK,TopSecretResponse(position=Coordinates(x=-400.0, y=300.0), message=este es un mensaje secreto),[]>
```

Documentación adicional

La documentación relacionada a **Javadoc** se encuentra en la ruta del proyecto:

/doc/index.html

Dentro del siguiente archivo se puede encontrar información relevante al funcionamiento del sistema:

/README.md

El proyecto se encuentra desplegado en la URL (incluyendo swagger):

https://quasarchallenge.uc.r.appspot.com/swagger-ui.html

Se puede importar la colección de postman desde el archivo:

/Challenge.postman_collection.json