

## **1)Tecnologias utilizadas**

- Java 8
- Spring Boot 2.7.16-Snapshot
- Spring Data
- MySQL 5.7.43
- Tomcat 9.0
- Apache Maven

## **2)Instruções para execução**

**A aplicação foi disponibilizada dentro de uma instância do Beanstalk da AWS Cloud, os exemplos de endpoints e a coleção do Postman enviados vão considerar o teste usando a referida instância, caso se queira testar na própria máquina, basta executar a aplicação na própria máquina e substituir a URL por localhost:8080.**

**Por exemplo, para ver o Swagger da aplicação basta entrar em:**

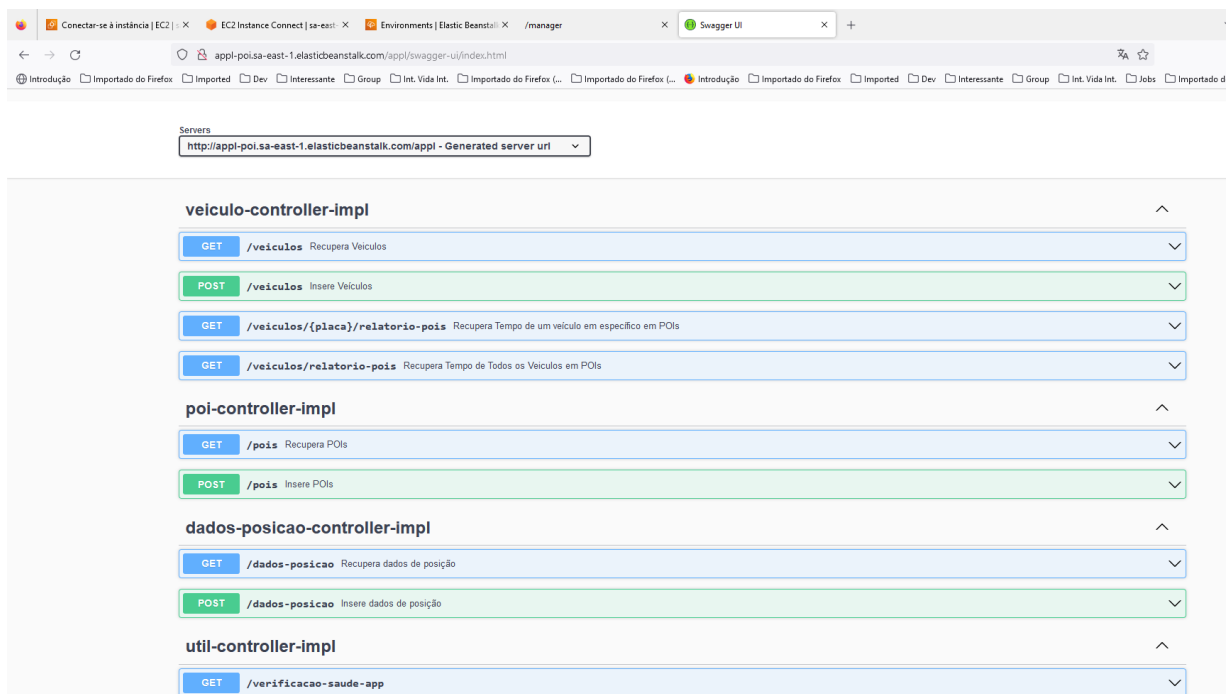
`http://appl-poi.sa-east-1.elasticbeanstalk.com/appl/swagger-ui/index.html`

**Caso se queira acessar o mesmo endpoint na própria máquina:**

`localhost:8080/appl/swagger-ui/index.html`

No repositório, dentro da pasta “/artefatos”, existe o arquivo appl.war já compilado pronto para ser executado como uma aplicação Java dentro de um servidor Tomcat 9.0. Como alternativa, pode-se executar a aplicação dentro de uma IDE de preferência importando o projeto no utilitário para importar projetos do tipo Maven e adicionar a aplicação dentro de um Tomcat 9.0 para executá-la. A aplicação ao ser iniciada já faz todo o processo de criação de banco de dados e tabelas (estrutura DDL) e já faz a inserção de dados das planilhas fornecidas como exemplo. Dentro da pasta raiz do projeto existem duas “collections” do Postman já que contém todos os endpoints configurados corretamente para interação com a aplicação, uma com os endpoints configurados para o Beanstalk da AWS e outra para teste em localhost.

Caso se faça o deploy em outro local que não seja o localhost basta informar o endereço e o endpoint “appl/swagger-ui/index.html” para acessar a interface Swagger e ver a documentação detalhada dos endpoints. Abaixo segue uma imagem amostra do Swagger da aplicação rodando no Beanstalk da AWS:



Dentro do arquivo application-dev.properties, é possível customizar a configuração de banco de dados entre outras propriedades definidas:

```

1 server.port=8080
2 debug=true
3 logging.level.org.springframework.boot.autoconfigure=ERROR
4
5 spring.datasource.url=jdbc:mysql://database-1.c8cisywhajhq.sa-east-1.rds.amazonaws.com:3306/db_appl_poi?createDatabaseIfNotExist=true
6 spring.datasource.username=admin
7 spring.datasource.password=9S1bH4RCreDx
8 spring.jpa.hibernate.ddl-auto=update
9 spring.jpa.defer-datasource-initialization=true
10 spring.sql.init.platform=mysql
11 spring.sql.init.mode=always
12 spring.sql.init.data-locations=classpath*:data_veiculos.sql,classpath*:data_dados_posicao.sql,classpath*:data_pois.sql
13
14
15 # database configs
16 spring.datasource.validationQuery=SELECT 1
17 spring.jpa.properties.hibernate.dialect.storage_engine=innodb
18 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL57Dialect
19 #spring.jpa.show-sql=true
20

```

### 3) Características e estrutura da aplicação

Foram criados três estruturas de CRUDs, uma para os “Dados de Posição”, outra para os “POIs” e outra para os “Veículos”, se pode recuperar e inserir essas três entidades nos endpoints correspondentes configurando a aplicação cliente de acordo com a documentação do Swagger, a “collection” do Postman criada já tem todos os endpoints configurados.

Para obter o relatório de tempo dos veículos nos POIs basta se usar um dos dois endpoints:

<http://appl-poi.sa-east-1.elasticbeanstalk.com/appl/veiculos/relatorio-pois>

ou

<http://appl-poi.sa-east-1.elasticbeanstalk.com/appl/veiculos/{placa}/relatorio-pois>

O primeiro recupera o tempo sem fazer filtragem de veículos, permitindo apenas a filtragem por POI e data desejada, o segundo recupera o tempo do veículo de placa passado como parâmetro na URL. Foi feito dessa forma para obedecer aos padrões RESTful de recuperação de informações.

Pode-se inserir novos Dados de Posição e POIs nos endpoints correspondentes para fins de escalabilidade e testes da aplicação.

Sobre o cálculo das coordenadas que pertencem ou não a um POI foi usado um algoritmo que executa a lógica da “aproximação equiretângular”, esse algoritmo foi escolhido dado que ele funciona bem com distâncias curtas e possui um bom desempenho e sua precisão é aceitável para a aplicação em questão. O site abaixo contém uma explicação muito boa sobre a fórmula e outros algoritmos para solucionar o problema:

<http://www.movable-type.co.uk/scripts/latlong.html>

Dessa forma são achados os intervalos de latitude e longitude que pertencem ao POI dado o raio, e posteriormente antes do computo do tempo que o veículo permaneceu no POI, são excluídos todos os pontos que não pertençam à circunferência do POI e somente pertençam ao quadrado que inscreve a circunferência do POI, eliminando assim a chance de algum “corner-case”.

Exemplo de resposta do primeiro endpoint com os dados de teste fornecidos:

```
[
{
  "poi": "CAR0012",
  "veiculoPOI": {
    "PONTO 1": "00h 00m 00s",
    "PONTO 10": "00h 00m 00s",
    "PONTO 11": "00h 00m 00s",
    "PONTO 12": "00h 00m 00s",
    "PONTO 13": "00h 00m 00s",
    "PONTO 14": "00h 00m 00s",
    "PONTO 15": "00h 00m 00s",
    "PONTO 16": "00h 00m 00s",
    "PONTO 17": "00h 00m 00s",
    "PONTO 18": "00h 00m 00s",
    "PONTO 19": "00h 00m 00s",
    "PONTO 2": "00h 00m 00s",
    "PONTO 20": "00h 00m 00s",
    "PONTO 21": "00h 00m 00s",
    "PONTO 22": "00h 00m 00s",
    "PONTO 23": "00h 00m 00s",
    "PONTO 24": "95h 39m 28s",
    "PONTO 3": "00h 00m 00s",
    "PONTO 4": "00h 00m 00s",
    "PONTO 5": "00h 00m 00s",
    "PONTO 6": "00h 00m 00s",
    "PONTO 7": "00h 00m 00s",
    "PONTO 8": "00h 00m 00s",
    "PONTO 9": "00h 00m 00s"
  }
}
```

```

},
{
  "poi": "TESTE001",
  "veiculoPOI": {
    "PONTO 1": "02h 50m 55s",
    "PONTO 10": "00h 00m 00s",
    "PONTO 11": "00h 00m 00s",
    "PONTO 12": "00h 00m 00s",
    "PONTO 13": "00h 00m 00s",
    "PONTO 14": "00h 00m 00s",
    "PONTO 15": "00h 00m 00s",
    "PONTO 16": "00h 00m 00s",
    "PONTO 17": "00h 00m 00s",
    "PONTO 18": "00h 00m 00s",
    "PONTO 19": "00h 00m 00s",
    "PONTO 2": "02h 50m 55s",
    "PONTO 20": "00h 00m 00s",
    "PONTO 21": "00h 00m 00s",
    "PONTO 22": "00h 00m 00s",
    "PONTO 23": "00h 00m 00s",
    "PONTO 24": "183h 12m 42s",
    "PONTO 3": "00h 00m 00s",
    "PONTO 4": "00h 00m 00s",
    "PONTO 5": "00h 00m 00s",
    "PONTO 6": "00h 00m 00s",
    "PONTO 7": "00h 00m 00s",
    "PONTO 8": "00h 00m 00s",
    "PONTO 9": "00h 00m 00s"
  }
}
]

```

#### 4) Possíveis melhorias futuras

1) Usar diferentes threads de execução no computo do tempo dentro das coordenadas do POI para mitigar o possível gargalo de desempenho no aumento do tamanho da entrada, pois no pior caso a complexidade assintótica do algoritmo construído é da ordem de  $O(n^3)$ .

#### 5) Observações

1) Toda vez que a aplicação estiver sendo iniciada, todos os dados de posição que são relacionados aos veículos que foram fornecidos na planilha “posições.csv”, e todos os dados de POIs que foram fornecidos na planilha “base\_pois\_def.csv” serão excluídos e inseridos novamente na base de dados com informações idênticas as contidas nas planilhas em questão.

2) Na base de dados fornecida foram encontradas algumas datas onde o dia da semana informado estava divergente do dia real da data como mostrado na imagem abaixo. Podemos ver, por exemplo, que o dia 13/12/2018 aconteceu numa quinta-feira e não em uma quarta-feira como informado na base de dados, dado esse problema foi considerado o dia real para todos os casos encontrados, porém esse fato não impactou no cálculo de tempo que os veículos passaram dentro dos POIs informados.

