Data prevista para entrega: 13/02/2023
Data entrega: -
Data correção: 18/04/2023

Ponto de Interesse

Deve ser desenvolvido um sistema para encontrar **pontos de interesse** de acordo com o que o usuário buscar.

Um ponto de interesse contém a coordenada X, coordenada Y e também um nome.
Nome: Não pode ser nulo ou vazio
Coordenada X: Não pode ser nulo e deve aceitar valores positivos ou negativos com ponto flutuante, exemplo: 22.000
Coordenada Y: Não pode ser nulo e deve aceitar valores positivos ou negativos com ponto flutuante, exemplo: -22.000

Alguns recursos devem ser desenvolvidos obrigatoriamente:

Inserir um ponto de interesse

POST -> */ponto-de-interesse*

- Deve ser inserido um ponto de interesse no qual será usado para realizar buscas pelo usuário.

Buscar todos os pontos de interesse

GET -> */ponto-de-interesse*

- Deve retornar todos os pontos de interesse cadastrados na base, se não existir nenhum deve ser retornado uma lista vazia.

Buscar ponto de interesse por nome

GET -> */ponto-de-interesse/{name}*

- Deve retornar o ponto de interesse que tem o nome cadastrado igual ao nome do parâmetro, se não for encontrado deve ser retornado uma **exceção** com uma mensagem apropriada.

Buscar ponto de interesse por filtros

GET -> */ponto-de-interesse/$params*

- Deve retornar os pontos de interesse baseado na posição do usuário e na distância que ele informar, então devemos receber as informações: **coordenada X**, **coordenada Y** e por último a **distância** (todos parâmetros são obrigatórios e não podem ser nulos), com esses parâmetros devemos encontrar todos os pontos de interesse entre a suas *coordenadas* + *distância* ou *coordenadas* - *distância*, se não

encontrar nenhum ponto de interesse baseado no filtro o retorno deve ser uma lista vazia.

Definições Técnicas

- A aplicação deve ser desenvolvida com Java + Spring boot (as versões ficam ao seu critério). Somente é necessário escrever um arquivo **README.md** com:
  - Tecnologias utilizadas no projeto e suas versões e as decisões para usá-las;
  - Como executar o projeto em uma nova máquina;
  - Os recursos expostos na fachada rest (se não utilizado Swagger);

- A aplicação deve conter uma fachada para REST UI para consumo e verificação de funcionamento, por exemplo: **Swagger** ou inserir a documentação no arquivo README.md de como utilizar no Postman.

- Deve ser desenvolvido pelo menos os testes unitários para a classe de serviço, link para estudo: **jUnit + Mockito**

- Deve ser criado uma pasta no root chamada **challenge** com a versão exportada em PDF deste arquivo.

- O projeto deve ser entregue em um repositório com o nome: **projeto-ponto-de-interesse** no Github e deve ser inserido o link abaixo.

Link do Github

Github: https://github.com/Guilherme-oliver/coordenadas

Correções e Melhorias

Definições Técnicas

- Foi escrito um README.md com muito texto e pouca formatação o que torna um pouco difícil de entender as decisões do projeto
- Foi inserido como executar os endpoints no Postman, porém tem maneiras melhores de se fazer: exportando a coleção do postman e colocando no projeto ou somente usando o Swagger (foi deixado na documentação).
- Não foi escrito nenhum teste na aplicação, novamente vou deixar o material para estudo: **jUnit + Mockito**
- Não foi criado uma pasta no root chamada **challenge** com esse arquivo em formato PDF
- Projeto não entregue com o nome que foi solicitado
- Projeto não entregue dentro do Prazo

Desafio

Prós

- Você fez muitos endpoints extras, isso é bom mas não foram pedidos.
- A distribuição das pastas dentro do main ficaram legais, uma dica seria usar os nomes em inglês.

Cons

- Todos os endpoints criados estão usando os objetos que são salvos no banco de dados como Request e como Response e isso não é uma boa prática, você viu isso no curso do Nélio porém eu deixarei aqui uma documentação falando mais.
- Todas as validações que você fez no objeto Coordenadas como ***@NotEmpty(message = "O nome não pode ser nulo")*** devem ser feitas no objeto de Request (CoordenadasRequest.java)
- O seu metodos POST ficou confuso, não tem a menor necessidade de utilizar isso:
    - *ServletUriComponentsBuilder.fromCurrentRequest().path("/{id}").buildAndExpand(objeto.getId()).toUri();;*
- O seu recurso inteiro de ***findByParams*** está com a lógica errada, e ela é bastante simples, o que você deve fazer é:
    - Receber os 3 parâmetros, isso você está fazendo;
    - Buscar todas as coordenadas do banco de dados (vai te retornar uma lista)
    - Com a lista recuperada você vai eliminar todas a coordenadas onde:
        - objetoRecuperadoDoBando.coordenadaX - parametroRecebido.coordenadaX <= parametroRecebido.distancia && objetoRecuperadoDoBando.coordenadaY - parametroRecebido.coordenadaY <= parametroRecebido.distancia

- Com isso você já vai ter todas as coordenadas que devem ser recuperadas para o usuário.