Desafio Técnico: Backend

Cenário

Uma empresa de transporte de grãos, com diversas filiais pelo Brasil, possui um parque de balanças para pesagem de caminhões carregados com grãos. Como parte da digitalização e otimização de seus processos, a empresa implementou um sistema baseado em ESP32, integrados a cada balança e com câmera LPR, responsáveis por enviar leituras de peso automaticamente para um servidor central da empresa. Porém, o sistema de estabilização das balanças ainda não foi aprimorado, resultando em oscilações constantes nos valores de peso reportados.

Neste contexto, a empresa precisa receber, processar e armazenar os dados das balanças de modo eficiente e confiável, possibilitando calcular custos e identificar oportunidades de lucro no transporte de grãos.

Sua missão é criar uma solução robusta para ingestão, estabilização e armazenamento das leituras de peso.

Premissas e regras do negócio:
- Cada caminhão parte da filial com a informação do tipo de grão que buscará em uma fazenda.
- No retorno à filial, o caminhão é direcionado para a doca e passa pela pesagem na balança.
- Enquanto houver um caminhão sobre a balança, o ESP32 enviará medições para o servidor da empresa a cada 100ms.
- Depois de descarregar, o caminhão pode receber uma nova demanda de transporte.
- Cada tipo de grão possui um preço de compra por tonelada.
- O preço de venda de cada tipo de grão é determinado aplicando uma margem de lucro, que deve variar entre 5% (mínima) e 20% (máxima) sobre o custo de compra.
- A margem de lucro é inversamente proporcional à quantidade disponível de cada tipo de grão na doca: quanto mais escasso o grão, maior a margem aplicada.

Detalhes do protocolo de comunicação das balanças:
- Cada balança (identificada pelo seu `id`) utiliza um ESP32 que envia, a cada 100ms enquanto houver caminhão presente, uma requisição HTTP ao servidor no seguinte formato:

```json
{
  "id": "<id da balança>",
  "plate": "<placa do caminhão>",
  "weight": <peso total>
}
```

> Observação: não é necessário implementar fisicamente a balança; para simular o ESP32, basta realizar chamadas HTTP ao endpoint responsável por receber os dados das pesagens.

Desafio

1. Cadastros
Implemente cadastros para armazenar:
- Caminhão, Tipo de grão, Filial, Balança, Transação de transporte
- Transação de transporte é a transação de compra e pesagem de grãos de um tipo para um caminhão, inicio e fim.

2. Recepção dos Dados das Balanças
Implemente um endpoint HTTP capaz de receber as requisições do ESP32 conforme o JSON acima (`plate`, `weight`). Considere que todas as balanças podem enviar dados simultaneamente.

3. Persistência com Estabilização
Salve os dados de pesagem **apenas quando o peso estiver estabilizado**. Elabore e descreva uma estratégia para identificar automaticamente o momento em que a balança está estabilizada.

Armazene:
- Placa
- Peso bruto estabilizado
- Tara (do cadastro)
- Peso líquido (bruto – tara)
- Hora e data da pesagem
- Id da balança
- Tipo de grão
- Custo da carga

4. Relatórios/Estatísticas
Implemente endpoints para fornecer (ao menos):
- Pesagens por filial, por caminhão, por tipo de grão, por período
- Custos de compra, por filial, por caminhão, por tipo de grão, por período

- Lucros possiveis, por filial, por caminhão, por tipo de grão, por período

Obs: Crie uma estratégia para que o frontend possa requisitar esses dados de forma fácil.

Essas informações ajudarão a otimizar a frota e identificar oportunidades de melhorias operacionais.

5. Extras (Diferenciais)
- Autenticação das balanças (valide se a requisição é de uma balança autorizada).
- Retentativa e idempotência nas requisições da balança.
- Deploy em nuvem, testes unitários e documentação OpenAPI/Swagger.
- Sugestões/Desenho da arquitetura
- Sugestão de como expandir e melhorar o processo.

Restrições Técnicas
- Utilize Node.js (Express/Nest.js), Java (Spring Boot) ou C#
- Solução pode ser entregue em uma API REST, com banco de dados relacional ou NoSQL.
- Não é necessário implementar a balança, a parte do ESP32 pode ser simulada chamando o endpoint HTTP para receber os dados.
- Containerização (Docker) é um diferencial.

O Que Esperamos?
- Código limpo, modular e bem documentado.
- README com: instruções de execução, arquitetura, decisões de estabilização.
- Scripts para criação do banco/tables/migrations.
- Testes básicos para alguns endpoints.