Documento de solução de software

Introdução

Você é um desenvolvedor em uma grande empresa do setor de óleo e gás. Uma das plantas dessa empresa instalou sensores em seus 2.000 equipamentos e você foi encarregado de criar a infraestrutura para receber os dados desses sensores em tempo real. Os equipamentos são capazes de enviar dados no formato JSON para um endpoint. Um exemplo do payload é:

{

"equipmentId": "EQ-12495",

"timestamp": "2023-02-15T01:30:00.000-05:00",

"value": 78.42

}

1 - Modele um banco de dados da sua escolha para o caso de uso apresentado;

2 - Crie uma API com um endpoint que receba as requisições em tempo real e armazene no banco de dados;

3 - Alguns dos sensores da planta podem apresentar falhas técnicas, resultando em lacunas nos dados. Para lidar com isso, o fornecedor pode enviar arquivos CSV com os dados perdidos. Adicione na API um endpoint que receba um arquivo CSV, realize o parser dos dados e salve os valores no banco de dados, o formato do CSV, segue o mesmo padrão do payload:

equipmentId; timestamp; value

EQ-12495; 2023-02-15T01:30:00.000-05:00; 78.42

4 - Crie uma tela que exiba o valor médio de cada sensor nas últimas 24 horas, 48 horas, 1 semana ou 1 mês. Sua tela deve possuir gráficos para facilitar a análise;

5 - Crie testes de integração e unitários para a solução;

6 - Crie scripts que automatizem o deploy e execução das aplicações;

7 - Implemente um mecanismo de autenticação no sistema;

8 - Crie um relatório de teste de carga para a solução num cenário de 500, 1000, 5000 e 10000 requisições simultâneas.

Abaixo está a exemplificação ponto a ponto de cada uma das tarefas acima:

Começando pela inicialização do APP

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Onde a função createApp é chamada,

Texto

Descrição gerada automaticamente

A função RouteRegistrer, inicializa as rotas que serão usadas

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente

1 - Modele um banco de dados da sua escolha para o caso de uso apresentado;

Para modelagem do banco, foi escolhido o banco postgre, e a ORM alchemy(flesk)

Flask==3.0.3

Flask-SQLAlchemy==3.1.1

Para subir o banco foi usado o Docker , com o configuração da yml abaixo:

Texto

Descrição gerada automaticamente

Na inicialização do banco de dados há um apontamento de um script sql para criação das tabelas

Texto

Descrição gerada automaticamente

Estão as modelagens de dados usadas na aplicação:

[Texto

Descrição gerada automaticamente](#_top)

Texto

Descrição gerada automaticamente

Vale ressaltar, que poderia ter sido usado um recurso da ORM chamado create.all.

A configuração da URL do alchemy está setada na inicialização do app no

Texto

Descrição gerada automaticamente

Inicialização do banco de dados:

Texto

Descrição gerada automaticamente

2 - Crie uma API com um endpoint que receba as requisições em tempo real e armazene no banco de dados;

Endpoint criado para atender essa demanda:

Rota: /sensor/data / argumento: POST

Tela de computador

Descrição gerada automaticamente

Função de validação do jason a ser inserido, sendo essa função um service que serve a tratativa de dados do sensor(caminho no comentário):

Texto

Descrição gerada automaticamente

Depois que os dados são validados pela função é feita uma tentativa de inserção de dados no banco de dados através das funções da ORM e da modelo de dados sensor

Tela de celular com aplicativo aberto

Descrição gerada automaticamente

É feito uma tratativa de try / exception para prevenção de crash no app

Caso os dados estejam corretos e a ORM consiga fazer a gravação de dados no banco sera retornado um jason mostrando o dado inserido e o código 201 – CREATED

Através da função personalizada createResponse:

Texto

Descrição gerada automaticamente

Que através da make\_response e jsonify, deixa o json é um formato valido;

3 - Alguns dos sensores da planta podem apresentar falhas técnicas, resultando em lacunas nos dados. Para lidar com isso, o fornecedor pode enviar arquivos CSV com os dados perdidos. Adicione na API um endpoint que receba um arquivo CSV, realize o parser dos dados e salve os valores no banco de dados, o formato do CSV, segue o mesmo padrão do payload;

equipmentId; timestamp; value

EQ-12495; 2023-02-15T01:30:00.000-05:00; 78.42

Endpoint criado para atender essa demanda:

Rota: /sensor/data/csv / argumento: POST

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Essa rota é a que contêm o maior número de validações, justamente para que possa ser identificado qualquer erro em qualquer sensor

Depois que as validações dos dados do sensor foram feitas, tentamos realizar o commit no banco de dados, em caso de exceção, devolvemos o erro.

Uma possível melhoria, seria ela criar uma alertar se um sensor desse erro repetidamente no envio dos dados, em uma tebela de erros gravar o equipamentId do sensor sempre que ele apresentar erro, e definir os error por códigos, assim teríamos uma visualização dos erros de cada sensor por id.

Nesse caso em caso de qualquer erro no envio do CSV, a função desvalida ele como um todo, dependendo do cenário, a função deve ser modificada para validar os dados individualmente e ignorar dados com erro;

4 - Crie uma tela que exiba o valor médio de cada sensor nas últimas 24 horas, 48 horas, 1 semana ou 1 mês. Sua tela deve possuir gráficos para facilitar a análise;

A base da tela Index.HTML veio de uma inteligência artificial

As funções que ela usa para alimentar os gráficos funcionam da seguinte forma:

Texto

Descrição gerada automaticamente

Através da ROTA: /sensor/data/average / argumento: GET

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

A rota chama uma classe para agrupar as informações de acordo com a data do timestamp

Texto

Descrição gerada automaticamente

A rota traz os dados agrupados:

Texto

Descrição gerada automaticamente

No caso acima não há dados no banco para exibir\*

Os dados são exibidos em formato de pizza na tela inicial:

Gráfico, Gráfico de pizza

Descrição gerada automaticamente

O botão geral exibi 3 gráficos sendo eles: últimas 24 horas, últimas 48 horas, última semana, o botão último mês, exibe somente o gráfico de dados do último mês, os dados estão agrupados por equipamentId

5 - Crie testes de integração e unitários para a solução;

Foram criadas algumas funções de test analisar a aplicação:

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Foi usado unittest para fazer funções de testes para rotas e services

Também é possível gerar um CSV de X linhas pela função payload\_csv

6 - Crie scripts que automatizem o deploy e execução das aplicações;

O deploy amantizado foi feito através do Docker, Docker-compose

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

7 - Implemente um mecanismo de autenticação no sistema;

A autentificação de usuário no sistema foi feita através do jwt e bcrypt usando as bibliotecas

flask\_bcrypt==1.0.1

PyJWT==2.9.0

Inicialização do modulo de autentificação:

Texto

Descrição gerada automaticamente com confiança média

Inserção de usuário:

Texto

Descrição gerada automaticamente

Validando o body da request através da função validBody

Texto

Descrição gerada automaticamente

A senha do usuário é criptografada através do bcrypt para ser inserida no banco

Para realizar o login é chamada a rota

Texto

Descrição gerada automaticamente

Que valida a usuário/senha enviada com usuário/senha que esta no banco

Para verificar se o usuário tem um token valido foi feito a rota protected

Texto

Descrição gerada automaticamente

Para renovar o token do usuário foi feita a rota refrash

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

8 - Crie um relatório de teste de carga para a solução num cenário de 500, 1000, 5000 e 10000 requisições simultâneas.

Um relatório de teste foi criado e esta na pasta tests.