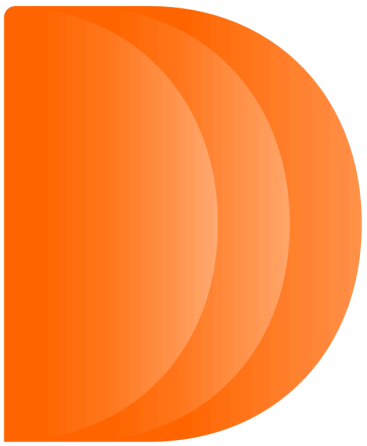

dtLabs

Tecnologias que avançam a humanidade

AVALIAÇÃO PRÁTICA - ESTÁGIO DE BACKEND

RECADO AO CANDIDATO

Toda a equipe da **dtLabs** gostaria de te parabenizar pela conquista de ter chegado até aqui! Acreditamos que, independentemente do resultado desta avaliação, possamos manter contato para uma possível futura oportunidade de trabalho em conjunto.

De todos os nossos membros,

Uma boa prova!

INSTRUÇÕES DA AVALIAÇÃO

Este exame tem como intuito avaliar suas competências referentes a:

- Desenvolvimento de APIs.
- Desenvolvimento de testes.
- Lógica de programação.
- Engenharia de Software.

FORMATO DE ENTREGA

1. Crie uma conta no GitHub, caso você não tenha.
2. Crie um repositório chamado **exame-backend-dtlabs-2025**. O repositório deve ser público.
3. Desenvolva o exame neste repositório, ele será utilizado para avaliação.
4. O **README.md** do repositório deve conter as instruções necessárias para testar a aplicação.
5. Fique a vontade para usar arquivos **Makefile**, por exemplo, para automatizar alguns processos. E use quaisquer outras ferramentas que você julgar necessário. O que iremos avaliar é a aplicação, testes que você desenvolveu, padrões de projeto aplicados e se o que foi pedido foi entregue.

Após a finalização, envie um e-mail com o assunto **PROVA BACKEND 2025** para o e-mail:

marismar.malara@dt-labs.ai

PRAZO PARA ENTREGA

Após receber este documento, você possui o prazo de 7 (sete) dias para resolver o *case* proposto.

OBSERVAÇÕES

1. O *case* **deve** ser desenvolvido em Python utilizando a framework FastAPI.
    a. **Não serão aceitas resoluções em outro framework que não seja FastAPI ou escrito em outra linguagem que não seja Python.**
    b. **Você pode utilizar qualquer outra biblioteca que julgar necessário.**
2. Você deve utilizar o PostgreSQL como banco de dados. Não serão aceitas resoluções que utilizem outros bancos.
3. Para o banco de dados escolhido, **é obrigatório executá-lo em um Docker**. Não serão aceitas resoluções que não utilizem o Postgre dentro de um Docker.
4. Você deve criar um repositório em seu Github e deve deixá-lo público. Você deve criar um arquivo **README.md** detalhando como executar o repositório, pois o avaliador irá testá-lo.
5. Recomenda-se o uso de Docker para executar a aplicação final.
6. Você **deve** criar testes para a sua aplicação.
7. Para os testes, é obrigatório o uso da biblioteca **PyTest.**
8. Você pode criar um docker-compose.yaml para subir a aplicação completa.
9. Evite nomenclatura de variáveis e comentários em português.
10. Você pode escolher qualquer ORM, ou Query Builder, ou inclusive criar as queries manualmente. Fica a seu critério.
11. Caso você seja chamado para a entrevista técnica, parte da entrevista se destinará a você explicar toda a aplicação.
12. Caso deseje, pode utilizar um sistema de Caching, como Redis, MemCached e etc…
13. Não se preocupe com *migrations*, não queremos avaliar isso.
14. A resolução do *case* deve ser 100% RESTFul.
15. Você pode estruturar o banco de dados com qualquer plugin do PostgreSQL que julgar necessário, como TimescaleDb, Pgcrypto e etc…
16. Caso julgue necessário, pode usar um serviço de fila, como Redis, RabbitMQ, Kafka e similares.
17. Seu *case* será executado pelo avaliador em um computador bom o suficiente (ao menos esperemos que sim). Logo, é importante que as variáveis de ambiente estejam em um arquivo para que o avaliador possa reproduzir o ambiente. Ou, indicando como criar as variáveis de ambiente, através de um .env e etc…

CRITÉRIOS DE AVALIAÇÃO

- Velocidade de entrega do teste resolvido.
- Qualidade dos testes.
- Estrutura da aplicação.
- Coerência com o que foi solicitado.
- Tratamento de erros.
- Status Code coerentes com a aplicação.
- Documentação da API.
- Uso de Design Patterns.
- Uso correto dos verbos HTTP e Headers.
- Design do Banco de Dados.
- Documentação do repositório.

CASE

Você irá desenvolver o *backend* de uma aplicação de IoT. Seu produto consiste em um servidor que está localizado on-premise no seu cliente. Este servidor coleta dados de diversos sensores. Os servidores enviam para um único banco de dados. Cada servidor comporta até 4 (quatro) sensores diferentes:

- Sensor de Temperatura.
    - Valores são medidos em graus celsius.
- Sensor de Umidade.
    - Valores são medidos em %, de 0 a 100.
- Sensor de Tensão Elétrica.
    - Valores são medidos em Volts.
- Sensor de Corrente Elétrica.
    - Valores são medidos em Ampère.

É possível que um servidor tenha um sensor de temperatura e um sensor de umidade. Portanto, eles enviam os dois valores na mesma requisição. Cada servidor vai possuir apenas 1 (um) sensor de cada. Logo, não existem servidores que possuem 3 (três) sensores de temperatura e 1 (um) sensor de corrente elétrica.

Os servidores podem enviar dados com uma frequência de, no mínimo, 1 Hz, e, no máximo, 10 Hz.

A seguir, serão descritos os *endpoints* necessários para serem implementados e descritivo do que eles devem fazer.

Funcionalidades

1. Autenticação (JWT)

O sistema deve ter um mecanismo de autenticação baseado em JWT para proteger endpoints privados. Os servidores e usuários autenticados devem utilizar um token para acessar as funcionalidades restritas.

Endpoints esperados:

- `POST /auth/register` → Criar um novo usuário.
- `POST /auth/login` → Autenticar usuário e retornar um token JWT.

2. Registro de Dados dos Sensores

Os servidores on-premise devem ser capazes de enviar leituras dos sensores para a API.

Endpoint esperado:

- `POST /data`
  - Não requer autenticação por JWT.
  - Payload esperado:

```
Unset

{
  "server_ulid": "01JMG0J6BH9JV08PKJD5GSRM84",
  "timestamp": "2024-02-19T12:34:56Z",
  "temperature": 25.5,
  "humidity": 60.2,
  "voltage": 220.0,
  "current": 1.5
}
```

Regras:

- Os valores dos sensores são opcionais, mas pelo menos um deles deve ser enviado na requisição.
- O servidor deve verificar se `server_ulid` já existe antes de registrar dados.
- O timestamp deve estar no formato ISO 8601.

3. Consulta de Dados

O sistema deve permitir a consulta de dados armazenados, filtrando por intervalo de tempo, servidor e tipo de sensor. Além disso, o usuário pode solicitar dados agregados por minuto, hora ou dia, onde a agregação é feita calculando a média dos valores dentro do intervalo especificado.

Endpoint esperado:

- `GET /data`
  - Query Parameters:
    - `server_ulid` (opcional) → Filtra por um servidor específico.
    - `start_time` e `end_time` (opcional) → Intervalo de tempo.
    - `sensor_type` (opcional) → Exemplo: `temperature`, `humidity`.a
    - `aggregation` (opcional) → Define a granularidade da agregação de dados. Valores possíveis: `minute`, `hour`, `day`.
  - Requer autenticação por JWT.
  - Resposta esperada:

```
Unset

[
  {
    "timestamp": "2024-02-19T12:34:00Z",
    "temperature": 25.3
  },
  {
    "timestamp": "2024-02-19T12:35:00Z",
    "temperature": 24.9
  },
]
```

Regras:

- Se `aggregation` for informado, a API deve calcular a média dos valores dentro do período especificado (minuto, hora ou dia) e retornar apenas um único valor por intervalo.
- Caso `aggregation` não seja informado, os dados são retornados conforme foram armazenados.

- Não se preocupe em implementar paginação.

4. Monitoramento da Saúde do Servidor

O sistema deve permitir verificar se um servidor está online ou não. Além disso, deve ter um outro *endpoint* para listar todos os servidores.

Endpoints esperados:

- `GET /health/{server_id}`
  - Retorna um status baseado no último dado recebido do servidor.
  - Um servidor é considerado offline se não enviar dados há mais de 10 segundos.
  - Requer autenticação por JWT.
  - Resposta esperada:

```
Unset
{
  "server_ulid": "01JMG0J6BH9JV08PKJD5GSRM84",
  "status": "online",
  "server_name": "Dolly #1
}
```

- `GET /health/all`
  - Idêntico ao endpoint anterior, porém retorna a lista de todos os servidores cadastros daquele usuário.
  - Requer autenticação por JWT.
  - Resposta esperada:

```
Unset
{
  [
    {
      "server_ulid": "01JMG0J6BH9JV08PKJD5GSRM84",
      "status": "online",
      "server_name": "Dolly #1"
```

```
    },
    {
      "server_ulid": "01JMG3HQZPWQV0MN8GBSM9QDHZ",
      "status": "offline",
      "server_name": "Dolly #2"
    }
  ]
}
```

5. Registro de Servidores

Os servidores on-premise devem ser capazes de enviar leituras dos sensores para a API.

Endpoint esperado:

- `POST /servers`
  - Requer autenticação por JWT.
  - Payload esperado:

```
Unset

{
  "server_name": "Dolly #1",
}
```

Regras:

- A criação do ULID do servidor deve ser feita pelo backend.