Processo seletivo backend 2019 - Teste prático

Teste | Documentação

Autor: Gabriel Sá

- GitHub
- Linkedin

Teste

O objetivo deste teste é avaliar o seu conhecimento técnico em relação às ferramentas utilizadas no backend da Laura. Os serviços do sistema tem sua comunicação majoritariamente baseada em APIs, desenvolvidas utilizando frameworks e micro frameworks. A comunicação e a arquitetura do banco de dados também são pontos importantes para questões de performance e escalabilidade.

Neste teste, solicitamos que você defina uma arquitetura para as collections de um banco MongoDB e crie uma aplicação em python com API para gerenciamento dos dados nele presente. Lembre-se de seguir as melhores práticas de desenvolvimento.

Ainda que, por qualquer motivo, você não consiga completar este teste, recomendamos que ainda assim nos encaminhe o que foi desenvolvido. A falta de cumprimento de alguns dos requisitos aqui descritos não implica necessariamente na desconsideração do candidato. Em caso de dúvidas não hesite em entrar em contato.

Parte 1. Importar dados de “dataset_estudantes.csv” para o banco de dados

Utilizando a biblioteca pymongo, crie uma base de dados e uma collection, bem como os indíces que achar pertinente, contendo os dados do arquivo “dataset_estudantes.csv”. As credenciais da conexão do banco (host, porta, etc) e o nome da base de dados e da collection deverão ser lidas de variáveis de ambiente. O formato dos dados no banco de dados fica a seu critério, bem como o nome das variáveis de ambiente. O código gerado deverá estar em um arquivo import_data.py. Também solicitamos que nos seja enviado um exemplo do arquivo contendo as variáveis de ambiente como o nome env_file.env (solicitamos esse arquivo apenas para referência, podendo inclusive ser o que você usou localmente para testar).

Parte 2. Criar uma aplicação python com pymongo e Flask

A seguir são listados algumas definições a respeito dos endpoints que devem estar presentes na aplicação. Além do código, você também será responsável por definir o tipo de requisição HTTP de cada um dos endpoints (com exceção do primeiro), bem como seus status de retorno. O código da aplicação deverá estar em arquivo de nome server.py. Na aplicação, a API deve ser feita da seguinte maneira:

1. Listar todos os itens de uma modalidade em um período ordenados por data
    i. Tipo da requisição: GET
    ii. Parâmetros: modalidade, data de início e data de fim
    iii. Retorno: lista de todos os itens com modalidade, filtrando pelo período passado e ordenando de forma decrescente pela data dos documentos.
2. Listar todos os cursos de um campus
    i. Tipo da requisição: [a definir]
    ii. Parâmetros: campus
    iii. Retorno: lista de cursos do campus
3. Descobrir número total de alunos num campus em um dado período
    i. Tipo de requisição: [a definir]
    ii. Parâmetros: campus, data de início e data de fim
    iii. Retorno: número de alunos do campus no período
4. Cadastrar alunos
    i. Tipo da requisição: [a definir]
    ii. Parâmetros: nome, idade_ate_31_12_2016, ra, campus, município, curso, modalidade, nivel_do_curso, data_inicio
    iii. Retorno: sucesso/erro
5. Buscar aluno
    i. Tipo da requisição: [a definir]
    ii. Parâmetro: ra
    iii. Retorno: todos os dados do aluno
6. Remover aluno do banco
    i. Tipo da requisição: [a definir]
    ii. Parâmetros: ra, campus
    iii. Retorno: sucesso/erro

> Obs: o retorno de cada requisição deve ser um JSON válido.

Parte 3. Desenvolver cache interno

Desenvolver um cache simples, em memória, para que não seja necessária uma nova consulta no

banco de dados para os alunos recém-acessados. O cache deverá conter no MÁXIMO 10 itens (ou seja, dados de no máximo 10 alunos). O cache também deverá levar em conta dados recém-cadastrados. Exemplos de comportamento do cache:

1. O endpoint 5 (Buscar aluno) busca por aluno de ra 123. Em seguida, o mesmo endpoint é acessado também para o aluno de ra 123. Como esse dado já havia sido buscado no banco recentemente, a aplicação não deve fazer uma nova leitura no banco mas sim ler do cache;
2. O endpoint 4 (Cadastrar aluno) registra aluno de ra 321. Em seguida o endoint 5 (Buscar aluno) é acessado para o aluno de 321. Como esse aluno acabou de ser registrado, a aplicação não deve fazer nova leitura no banco mas sim ler do cache.

O critério de evasão (momento em que um dado deve ser removido da cache para dar lugar a outro mais recente) fica por sua conta.

Parte 4: Documentação da APIs

Desenvolva alguma forma de documentação das APIs.

Links para referência:

- Virtual Env
- Flask Docs
- Mongo API
- Mongo Docs

Documentação

Requisitos

Requisitos necessários para a instalação e execução do projeto.

- Python 3.8.3
- Virtualenv 20.0.25

Instalação

> ***NOTA:*** Repositório privado

```
git clone https://github.com/gabrielhjs/Laura_teste.git
cd Laura_teste
```

> ***NOTA:*** Crie uma máquina virtual

```
virtualenv venv_laura
. venv_laura\scripts\activate
pip install -r requirements/development.txt
```

Execução

```
python server.py
```

Banco de dados

A estrutura do banco de dados contém uma Collection chamada "students" que contém todos os dados da aplicação. O ra de cada estudante é único. Os cursos de cada estudante são armazenados na lista "courses".

Formato dos dados:

```
{
    "_id": {
        "$oid": "5f2e278ffb12be63b5a3f3e8"
    },
    "name": "GABRIEL SÁ",
    "age": 24,
    "ra": 8740.0,
    "courses": [
        {
            "campus": "TL",
            "county": "Três Lagoas",
            "course": "TÉCNICO EM ELETROTÉCNICA",
            "modality": "PRESENCIAL",
            "level": "INTEGRADO",
            "start_date": {
                "$date": 1430524800000
            }
        },
        {
            "campus": "TL",
            "county": "Três Lagoas",
            "course": "ELETRÔNICA",
            "modality": "PRESENCIAL",
            "level": ".FICCAMPUS",
            "start_date": {
```

```
                "$date": 1480809600000
            }
        },
        {
            "campus": "AQ",
            "county": "Aquidauana",
            "course": "TÉCNICO EM INFORMÁTICA",
            "modality": "PRESENCIAL",
            "level": "SUBSEQUENTE",
            "start_date": {
                "$date": 1596844800000
            }
        }
    ]
}
```


Endpoints

1. Busca por cursos de uma modalidade: `GET /api/school/modality/<modalidade>/<data inicial>/<data final>/`

Exemplo de resultado:

```
GET /api/school/modality/PRESENCIAL/2015-07-27/2015-07-27/
```


```
[
    {
        "name": "ADALTO SEBASTIAO DA SILVA JUNIOR",
        "age": 18.0,
        "ra": 8570.0,
        "courses": {
            "campus": "AQ",
            "county": "Aquidauana",
            "course": "TÉCNICO EM INFORMÁTICA",
            "modality": "PRESENCIAL",
            "level": "SUBSEQUENTE",
            "start_date": {
                "$date": 1437955200000
            }
        }
    },
    {
        "name": "ADRIANO COSTA BISPO DOS SANTOS",
        "age": 29.0,
        "ra": 13463.0,
        "courses": {
            "campus": "AQ",
```

```
            "county": "Aquidauana",
            "course": "TÉCNICO EM INFORMÁTICA",
            "modality": "PRESENCIAL",
            "level": "SUBSEQUENTE",
            "start_date": {
                "$date": 1437955200000
            }
        }
    },
    {
        "name": "ALINE MARIE RONDON TOSCANO DE BRITO GOMES",
        "age": 17.0,
        "ra": 6891.0,
        "courses": {
            "campus": "AQ",
            "county": "Aquidauana",
            "course": "DESENHISTA DE MÓVEIS",
            "modality": "PRESENCIAL",
            "level": ".FIC",
            "start_date": {
                "$date": 1437955200000
            }
        }
    },
   "continua..."
]
```


2. Busca todos os cursos de um campus: `GET /api/school/campus/<campus>/`

Exemplo de resultado:

```
GET /api/school/campus/tl/
```


```
[
    {
        "_id": "TL",
        "courses": "ANÁLISE E DESENVOLVIMENTO DE SISTEMAS"
    },
    {
        "_id": "TL",
        "courses": "AUTOMAÇÃO INDUSTRIAL"
    },
    {
        "_id": "TL",
        "courses": "ELETRÔNICA"
    },
```

```
    {
        "_id": "TL",
        "courses": "ESPANHOL BÁSICO"
    },
    {
        "_id": "TL",
        "courses": "ESPECIALIZAÇÃO EM DOCÊNCIA PARA A EDUCAÇÃO PROFISSIONAL, CIENTÍFICA E TECN
    },
    {
        "_id": "TL",
        "courses": "GRUPO DE ROBÓTICA"
    },
    "continua..."
]
```


3. Busca pela quantidade de alunos em um campus em um período: `GET /api/school/campus/students/<campus>/<data inicial>/<data final>/`

Exemplo de resultado:

```
GET /api/school/campus/students/tl/1015-07-27/3015-07-27/
```


```
[
    {
        "_id": "TL",
        "count": 836
    }
]
```


4. Cadastro de aluno: `POST /api/school/student/new/`

> Obs: Caso o ra do aluno já exista no banco de dados, os dados serão atualizados e o curso será adicionado. Caso contrário, o aluno será cadastrado no banco com o curso inserido.

Sucesso:


```
{
    "message": "Operation performed successfully",
    "status": "success"
}
```


Erro:

```
[
    {
        "error": "This field is required.",
        "field": "ra"
    }
]
```


5. Busca por dados de um aluno: `GET /api/school/student/<ra>/`

Exemplo de resultado:

```
GET /api/school/student/19023/
```


```
{
    "_id": {
        "$oid": "5f2e278ffb12be63b5a3d7fe"
    },
    "name": "ADRIANA TEIXEIRA SERRA",
    "age": 20.0,
    "ra": 19023.0,
    "courses": [
        {
            "campus": "AQ",
            "county": "0",
            "course": "OPERADOR DE COMPUTADOR",
            "modality": "EAD",
            "level": ".FIC",
            "start_date": {
                "$date": 1465516800000
            }
        }
    ]
}
```


6. Remover curso de aluno e/ou aluno: `POST /api/school/student/delete/`

> Obs: Caso o ra do estudante buscado tenha apenas o curso buscado o aluno será deletado do banco de dados. Caso contrário, apenas o curso será deletado da lista de cursos do aluno.

Sucesso:


```
{
    "message": "Operation performed successfully",
```

```
    "status": "success"
}
```

Erro:

```
{
    "message": "Data not found",
    "status": "error"
}
```