MyMusic

Dinâmica de trabalho:

Os primeiros 30 minutos serão utilizados para alinhar o desafio e estruturar as tarefas entre o grupo (planning). Na sequência o grupo se dividirá para desenvolver as tarefas planejadas durante 3 horas. No final, nos últimos 30 minutos, faremos uma *retrospectiva* onde cada participante poderá expor a sua solução e dificuldades encontradas.

Introdução:

Um cliente possui uma aplicação **.NET** para controle de músicas favoritas de seus usuários. Esta é uma aplicação legada onde o usuário pode interagir com sua lista de músicas favoritas, além disso ele possui um banco de dados que armazena estas informações e um serviço de APIs que realizam a integração deste cliente com o banco de dados.

Após diversas análises de desempenho foi identificado que existe um gargalo na aplicação legada, principalmente na camada de APIs sendo que a mesma está em um servidor monolítico que não comporta a demanda atual e também não permite o fácil e rápido dimensionamento de servidores para atender à demanda.

Foi então solicitado que vocês desenvolvam novos serviços para substituir a camada de APIs utilizando o banco de dados existente, e esta aplicação deve seguir alguns conceitos básicos. **A aplicação do cliente não será disponibilizada, somente o banco de dados.**

- **Ser construída utilizando técnicas de micro-serviços**, permitindo um dimensionamento independente de APIs, agrupadas acordo com os domínios de negócios identificados abaixo.
    - Planeje como o deploy da aplicação será feito (12factor);
- **Ser documentada**, Possuir página de documentação de APIs utilizando algum framework consolidado.
    - Utilize boas práticas de APIs REST e organização/nomenclatura do código.
- **Manter compatibilidade legado**, utilizar o banco de dados disponibilizado que contém todas as informações do legado.
- **Qualidade de código**, Pensar numa maneira de garantir a qualidade do código sendo que a intenção é evoluir as rotas existentes com certa frequência.

Narrativa de Negócio:

As funcionalidades básicas da aplicação que acessam as APIs compreendem em 5 ações principais descritas abaixo

- Permitir o usuário buscar novas músicas no banco de dados:
    1. O serviço deve validar se usuário informou ao menos 3 caracteres, retornando um HTTP 400 caso a consulta tenha menos de 3 caracteres.
    2. A busca deve ser realizada tanto nas coluna de nome de artista e nome da música.
    3. A busca por música não deve ser *case sensitive*.
    4. A busca deve retornar valores **contendo** o filtro, não necessitando de ser informado o nome completo de música ou artista.
    5. O retorno deve estar ordenado pelo nome do artista e depois pelo nome da música.
- Permitir o usuário informar o nome do usuário para buscar sua playlist:

1. O usuário deve informar um nome de usuário válido, retornando um 404 caso não encontre.
2. A busca por usuário deve ser *case sensitive.*

- Permitir o usuário escolher as músicas do resultado da busca que deseja adicionar na sua playlist:
    1. Deve receber um request contendo o identificador da música e o identificador da playlist.
    2. Deve validar se o identificador da música e o identificador da playlist existem.
- Permitir o usuário remover músicas de sua playlist:
    3. Deve receber um request contendo o identificador da música e o identificador da playlist.
    4. Deve validar se o identificador da música e o identificador da playlist existem.

Narrativa Técnica:

Multi-plataforma:

- Deverá ser utilizado **JAVA 8, Spring Boot e Maven**. A escolha da **IDE** é livre e fica a cargo do candidato.

Micro-serviços:

- Existem dois domínios de negócios que foram identificados e suas APIs devem ser construídas permitindo a possibilidade de serem "levantadas" em servidores de forma independente.
    1. **Músicas - Busca de Músicas**: Este domínio deve conter API de busca de Músicas
    2. **Playlist - Controle de Playlist**: Este domínio deve conter APIs de busca e alterações de playlist.

- API de listagem de Musicas:
    - Método: **GET**
    - Rota: **`/api/musicas?filtro='Bruno+Mars'`**
    - Parâmetros: **`QueryString: {filtro} string - Opcional`**
    - Retorno: **`Array do objeto "Musica" do modelo Json`**
    - Erros tratados: **204** com array vazio quando não houver dados e **400** quando caracteres de busca menor que 3
- API de Playlist de Usuário
    - Método: **GET**
    - Rota: **`/api/playlists?user='Robson'`**
    - Parâmetros: **`QueryString: {user} string - Obrigatorio`**
    - Retorno: **`Objeto "Playlist" do modelo Json`**
    - Erros tratados: **204** quando não encontrar usuário
- API de Adicionar relação de Musicas na Playlist
    - Método: **PUT**
    - Rota: **`/api/playlists/{playlistId}/musicas`**
    - Parâmetros:
        - **`Url: Path param: {playlistId} string`**
        - **`Body: Array do objeto "Musica" do modelo Json`**
    - Retorno: **200 OK**
    - Erros tratados: **400** quando identificadores não forem encontrados no banco.
- API de Remover relação de Músicas da Playlist
    - Método: **DELETE**
    - Rota: **`/api/playlists/{playlistId}/musicas/{musicaId}`**
    - Parâmetros:
        - **`Url: {playlistId}`**
    - Retorno: **200 OK**
    - Erros: **400** quando identificadores não forem encontrados no banco.

Banco de dados:

- O banco de dados é um SQLite e está localizado no diretório: **`"./MyMusic.db"`**

Documentação:

- As APIs devem estar documentadas na home de cada API..

Desempenho:

- Na API de GET de músicas, pensar numa maneira de guardar pesquisas pelo mesmo termo por 10min.

Modelo de Dados

JSON:

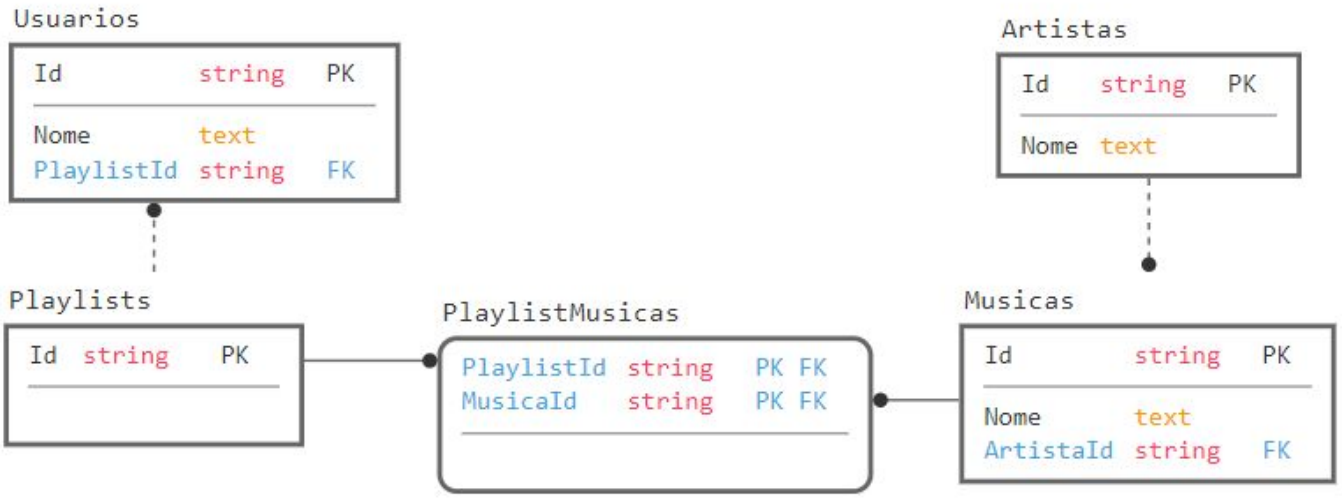
- Objeto **“Musica”**

```
{
  "id": "string",
  "nome": "string",
  "artistaId": "string",
  "artista": {
    "id": "string",
    "nome": "string"
  }
}
```

- Objeto **“Playlist”**

```
{
  "id": "string",
  "playlistMusicas": [
    {
      "playlistId": "string",
      "musicaId": "string",
      "musica": {
        "id": "string",
        "nome": "string",
        "artistaId": "string",
        "artista": {
          "id": "string",
          "nome": "string"
        }
      }
    }
  ],
  "usuario": {
    "id": "string",
    "nome": "string",
    "playlistId": "string"
  }
}
```

BANCO:



ATENÇÃO: Os campos Id que utilizam GUID (pode mapear como **string** devido à complexidade na compatibilidade com o UUID nativo do Java)

Dica:
Não é necessário, porém é possível utilizar uma ferramenta para abrir e visualizar o arquivo **MyMusic.db** de maneira mais fácil, como:https://sqlitestudio.pl/index.rvt

Será disponibilizada uma coleção do postman com requests de exemplo já montados, não é necessária sua utilização, é possível validar a implementação de outras maneiras, fica a cargo do time a sua utilização.

A coleção está no diretório **“`./MyMusic.postman_collection.json`”**, para utilizá-la baixar o plugin do postman: https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdggehcddcbncdddomop

Após instalação, importar a coleção arrastando para o postman, conforme imagem: