

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 2 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. Outro ponto que identificado durante a análise foi que a camada de apresentação precisava ser reformulada, para resolver problemas de usabilidade e ter uma aparência mais moderna.

Foi então solicitado que vocês desenvolvam novos serviços para substituir a camada de APIs utilizando o banco de dados existente e uma nova camada de apresentação. **A aplicação do cliente não será disponibilizada, somente o banco de dados.**

Requisitos:

- **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 APIs e a camada de apresentação.

Narrativa de Negócio:

As funcionalidades básicas da aplicação 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, a API deve retornar um HTTP 400 caso a consulta tenha menos de 3 caracteres e a camada de apresentação informar o usuário.
 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, caso contrário a API deve retornar um 404 caso não encontre e a camada de apresentação informar o usuário
 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:



Camada de apresentação:

O diagrama ilustra a interface 'My Music' com os seguintes elementos:

- Barra de Navegação:** Contém ícones para voltar, avançar, fechar e ir para casa, além de uma barra de endereço e um campo de busca.
- Formulário de Busca:** Possui dois campos: 'Music or artist:' com o valor 'Mus' e 'User:'.
- Grid de Músicas:** Uma tabela com duas colunas principais: 'Music name' e 'Artist name'.

Music name	Artist name	Ação
Music 1	Artist A	<input type="checkbox"/>
Music 2	Artist A	<input checked="" type="checkbox"/>
Music 3	Artist B	<input checked="" type="checkbox"/>
Music 4	Artist C	<input type="checkbox"/>
- Botões de Ação:** Botões '>>' e '<<' permitem mover músicas entre o grid de resultados e a playlist.
- Grid de Playlist:** Uma tabela similar ao grid de músicas, mas com uma terceira coluna para seleção.

Music name	Artist name	Seleção
Music 2	Artist A	<input checked="" type="radio"/>
Music 3	Artist B	<input type="radio"/>

- Utilizar preferencialmente o framework JavaScript Angular versão 5
- Basicamente o funcionamento da tela é o seguinte:
 - O usuário digita o nome completo ou parcial da música e tecla Enter
 - A camada de apresentação deve acionar a API de listagem de música e preencher o datagrid com o resultado obtido
 - O usuário tem a opção de criar ou atualizar uma playlist, para isso basta informar um usuário válido, selecionar as músicas desejadas através do checkbox do datagrid de músicas e acionar o botão 
 - A remoção de uma música da playlist pode ser feita através da seleção de uma música através do radio button e do acionamento do botão 

Camada de Backend:

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.

Modelo de Dados

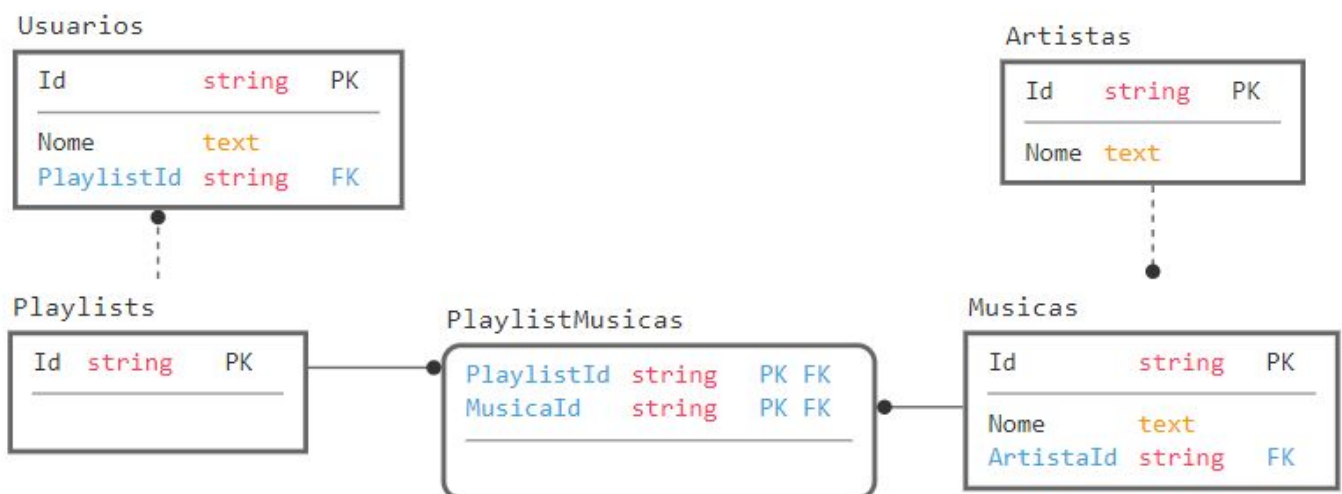
JSON:

- 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 mapear como **string** devido à complexidade na compatibilidade com o UUID nativo do Java.

Dicas:

- 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>
- Não é necessário desenvolver todas as camadas, por exemplo, front e back. O time pode decidir implementar somente as APIs ou somente uma funcionalidade, por exemplo, listagem de música front e back
- Para criar a base do projeto front executar o procedimento do Angular Cli: <https://github.com/angular/angular-cli#usage>
- Api exposta: <https://intense-ocean-93206.herokuapp.com/swagger-ui.html>