Introdução a APIS e Express.js gets

Exercícios Práticos com Express e Métodos HTTP

Instruções Gerais

Estes exercícios foram projetados para introduzir você ao framework Express e aos métodos HTTP. Eles são incrementais em dificuldade, começando desde a configuração básica de um projeto Express até a manipulação de recursos usando métodos HTTP como GET, POST, PUT e DELETE.

Requisitos

- Node.js e npm instalados

Configuração do Projeto em TypeScript

1. Inicie um novo projeto Node.js executando `npm init -y` no terminal.
2. Instale o TypeScript globalmente ou como uma dependência de desenvolvimento: `npm install -g typescript` ou `npm install --save-dev typescript`.
3. Instale o ts-node-dev `npm i ts-node-dev`
4. Crie um arquivo `tsconfig.json` para configurações do TypeScript. Você pode gerar um padrão executando `npx tsc --init`.
5. Instale os tipos do Node.js: `npm install --save-dev @types/node`.
6. Instale o Express e seus tipos: `npm install express @types/express`.
7. Instale o Express e seus tipos: `npm install -D @types/express`.
8. Instale o cors e seus tipos: `npm i cors @types/cors`
9. Instale o cors e seus tipos: `npm i cors -D @types/cors`
10. Crie a pasta build e a pasta src
11. Crie o arquivo index.ts dentro da pasta src
12. Dentro do package.json altere os scripts para

```
"scripts": {
      "dev": "ts-node-dev ./src/index.ts",
   "start": "tsc && node ./build/index.js",
   "build": "tsc"
},
```

    1. O script `dev` funciona de maneira similar ao `start`, porém ele atualiza o servidor automaticamente quando você salva o arquivo (hot reload).
13. Você pode agora executar os scripts criados.

Exercícios

Importante: Para todos os exercícios abaixo, como ainda não aprendemos a conectar o Express a um banco de dados, você deve utilizar vetores (arrays) para simular o acesso a uma tabela no banco de dados. Considere os vetores como uma representação dos dados armazenados. Você pode iniciar um vetor com alguns dados fictícios para que os testes e as operações façam sentido.

Estrutura Inicial do Projeto

Para facilitar o seu trabalho, fornecemos um template básico com um vetor que simula o acesso a uma tabela de banco de dados. O vetor `users` contém dados fictícios que serão usados nos exercícios seguintes.

Arquivo `src/index.ts`

```
import express from 'express';
import cors from 'cors';

const app = express();
app.use(cors());
app.use(express.json());

// Vetor que simula uma tabela de usuários no banco de dados
const users = [
    { id: 1, name: 'Niltão', playlists: [1, 2] },
    { id: 2, name: 'Nick', playlists: [3] },
    { id: 3, name: 'Will I Am', playlists: [] }
];

// Vetor que simula uma tabela de playlists no banco de dados
const playlists = [
    { id: 1, name: 'Forróck', tracks: ['Foguete não tem ré', 'O golpe taí', 'Forrock das aranhas'] },
    { id: 2, name: 'Funk dos cria', tracks: ['Vida Louca', 'Deu Onda', 'Ela Só Quer Paz'] },
    { id: 3, name: 'K-pop', tracks: ['Dynamite', 'Lovesick Girls', 'Gee'] }
];


// Os exercícios virão aqui...

const PORT = 3003;
app.listen(PORT, () => {
    console.log(`Server is running on port ${PORT}`);
});
```

Dicas de Métodos de Vetor (Array Methods)

Para manipular os dados nos vetores, você usará bastante os métodos de array do JavaScript. Aqui estão alguns dos mais úteis para estes exercícios, com exemplos genéricos para você entender a lógica.

`.find()`

- **O que faz?** Percorre o vetor e retorna o **primeiro elemento** que satisfaz uma condição. Se nenhum elemento for encontrado, retorna `undefined`.

- **Quando usar?** Ideal para quando você precisa buscar um item único por um identificador, como um ID.
- **Exemplo:** Encontrar um produto específico em um catálogo.

```
const produtos = [
   { id: 10, nome: 'Teclado', preco: 150 },
   { id: 25, nome: 'Mouse', preco: 80 },
   { id: 31, nome: 'Monitor', preco: 900 }
];

const produtoEncontrado = produtos.find((produto) ⇒ produto.id === 25);
console.log(produtoEncontrado); // Saída: { id: 25, nome: 'Mouse', preco: 80 }
```

`.filter()`

- **O que faz?** Percorre o vetor e retorna um **novo vetor** contendo **todos os elementos** que satisfazem uma condição. Se nenhum elemento corresponder, retorna um vetor vazio `[]`.
- **Quando usar?** Perfeito para buscas que podem retornar múltiplos resultados, como filtrar produtos por uma categoria ou pesquisar por um termo.
- **Exemplo:** Filtrar todos os livros de um determinado gênero.

```
const livros = [
   { titulo: 'O Senhor dos Anéis', genero: 'Fantasia' },
   { titulo: 'Duna', genero: 'Ficção Científica' },
   { titulo: 'Crônicas de Nárnia', genero: 'Fantasia' }
];

const livrosDeFantasia = livros.filter((livro) ⇒ livro.genero === 'Fantasia');
console.log(livrosDeFantasia);
// Saída: [{...}, {...}]
```

`.findIndex()`

- **O que faz?** Similar ao `.find()`, mas retorna o **índice (a posição)** do primeiro elemento que satisfaz a condição. Se não encontrar, retorna `-1`.
- **Quando usar?** Útil quando você precisa da posição do elemento para depois modificá-lo ou removê-lo do vetor original (com métodos que veremos mais adiante).
- **Exemplo:** Encontrar a posição de um aluno em uma lista.

```
const alunos = ['Ana', 'Bruno', 'Carla', 'Daniel'];

const indiceDoBruno = alunos.findIndex((aluno) ⇒ aluno === 'Bruno');
console.log(indiceDoBruno); // Saída: 1
```

Exercícios

Importante: Para todos os exercícios, utilize os vetores `users` e `playlists` como sua fonte de dados.

Exercício 1: Hello, Express!

1. **Objetivo**: Criar um servidor Express que responde com "Hello, Express!" quando alguém acessa a raiz ("/").
2. **Instruções**:
   - Crie um arquivo chamado `index.ts`.
   - Importe o pacote Express e configure o servidor para escutar na porta 3003.
   - Adicione uma rota GET para a raiz ("/") que envia a resposta "Hello, Express!".

Exercício 2: Listando todos os usuários

1. **Objetivo**: Criar um endpoint que retorna uma coleção de dados em formato JSON.
2. **Instruções**:
   - Adicione uma rota GET para "/users" que envie a variável `users` como uma resposta JSON.

Exercício 3: Parâmetros de Rota

1. **Objetivo**: Aprender a usar parâmetros de rota para enviar informações específicas.
2. **Instruções**:
   - Adicione uma rota GET para "/users/:userId" que envie o usuário correspondente ao `userId` fornecido na URL.

Exercício 4: Listando todas as playlists

1. **Objetivo**: Reforçar o conceito de retornar uma coleção de dados.
2. **Instruções**:
   - Crie uma rota `GET` para `/playlists` que retorna o vetor `playlists`

Exercício 5: Buscando playlists por nome

1. **Objetivo**: Aprender a usar parâmetros de consulta (`query parameters`) para filtrar dados.
2. **Instruções**:
   - Crie uma rota `GET` para `/playlists/search`.
     - A rota deve aceitar um parâmetro de consulta chamado `name`. O endpoint deve retornar as playlists cujo nome contenha o valor de `name` (a busca não deve ser sensível a maiúsculas e minúsculas).