🎵 Atividade de Revisão – Catálogo de Músicas

🎯 Objetivo

Revisar o fluxo completo de uma API REST com **Express**, aplicando o padrão **MVC** e utilizando a estrutura de pastas:

```
src/
 ├ data/
 ├ models/
 ├ controllers/
 ├ routes/
 └ app.js
```

Os alunos devem **planejar primeiro no papel** o funcionamento da API (endpoints, regras e dados) e **só depois** implementar o código no VS Code.

🧩 Contexto

Você foi contratado para desenvolver uma API de gerenciamento de músicas para um aplicativo chamado **SoundTrack Organizer**.
A aplicação deverá armazenar e manipular um pequeno **catálogo de músicas**.

📋 Campos obrigatórios da entidade `Música`

Cada música deverá conter os seguintes campos:

Campo	Tipo	Descrição
`id`	number	Identificador único da música (gerado automaticamente)
`titulo`	string	Nome da música
`artista`	string	Nome do cantor ou banda
`ano`	number	Ano de lançamento
`genero`	string	Gênero musical (ex: rock, pop, sertanejo, jazz)
`duracao`	string	Duração no formato "mm:ss" (ex: "03:45")

🧠 Etapa 1 – Planejamento no papel

Antes de abrir o VS Code, **planeje e anote no caderno**:

1. ✍️ **Todos os endpoints REST** que farão parte do CRUD:

- Quais serão as rotas (`/musicas`, `/musicas/:id`, etc.)
- Quais métodos HTTP serão usados (`GET`, `POST`, `PUT`, `DELETE`)
- O que cada rota deve retornar

2. 💡 **Um endpoint especial (personalizado)** que não seja do CRUD, por exemplo:
   - Buscar músicas por gênero
   - Buscar músicas por artista
   - Listar apenas títulos das músicas
   - Filtrar músicas lançadas após determinado ano
   - Calcular duração total das músicas cadastradas
3. ☑️ **Pequenas validações** que devem ser feitas:
   - Todos os campos obrigatórios devem ser preenchidos
   - O ano deve ser maior que 1900 e menor ou igual ao ano atual
   - Não pode haver duas músicas com o mesmo título e artista
4. 🧱 **Fluxo de dados da aplicação:**

```
data → model → controller → routes → app
```

Explique no papel o que cada camada faz e como elas se conectam.

💻 Etapa 2 – Implementação

Após o planejamento, implemente o projeto seguindo o padrão **MVC**:

1. **`data/musicas.js`**
   - Crie um vetor `musicas` contendo alguns exemplos iniciais de músicas.
2. **`models/musicaModel.js`**
   - Crie uma classe `MusicaModel` com métodos estáticos para:
     - Listar todas as músicas
     - Buscar por ID
     - Criar nova música
     - Atualizar música existente
     - Deletar música
   - Lembre-se de gerar o `id` automaticamente (ex: `Date.now()`)
3. **`controllers/musicaController.js`**
   - Crie uma classe `MusicaController` com métodos para tratar as requisições HTTP
   - Utilize `try/catch` para tratar erros
   - Retorne mensagens claras (ex: “Música criada com sucesso”)
4. **`routes/musicaRoutes.js`**

   - Crie as rotas e conecte cada uma aos métodos do controller

5. `app.js`
   - Configure o servidor Express
   - Adicione `express.json()` e as rotas de música

📝 Entrega esperada

- Estrutura de pastas organizada dentro de `src/`
- Código funcional e testado no **Insomnia**
- Mensagens de erro e sucesso bem definidas
- Comentários explicativos nas principais funções
- **Planejamento no papel** entregue junto (foto ou folha)

💬 Dica

> Planeje antes de codar. Entender o fluxo **data** → **model** → **controller** → **routes** → **app** é essencial para dominar o desenvolvimento de APIs com Node.js e Express.
