Desafio backend 2

Desafio Backend Rails – Sistema de Biblioteca Digital

1. Descrição Geral

Desenvolver uma API RESTful para gerenciar uma plataforma de biblioteca digital. Usuários autenticados poderão cadastrar, buscar, visualizar e gerenciar diferentes tipos de materiais (livros, artigos, vídeos), associando-os a autores (pessoas ou instituições). O sistema deve ser seguro, bem validado, seguir boas práticas REST, conter documentação e testes. Considere diferenciais como deploy, documentação interativa e um endpoint GraphQL.

---

2. Requisitos Funcionais

2.1 Autenticação

- Autenticação via e-mail e senha.
- Apenas usuários autenticados podem cadastrar, atualizar ou remover materiais.

2.2 Materiais e Tipos

- Cadastro de diferentes tipos de materiais: livros, artigos, vídeos, etc.
- Cada material deve conter campos genéricos e campos específicos do tipo.
- Todo material deve estar associado a um autor (pessoa ou instituição).
- O sistema deve permitir cadastro e vínculo de autores a materiais.

2.3 Permissões de Acesso

- Cada material só pode ser alterado ou removido pelo usuário que o cadastrou.
- Usuários podem visualizar materiais públicos de outros usuários.
- Implementar regras de autorização para proteger rotas sensíveis.

2.4 Status

- Todo material deve possuir um campo `status` com valores possíveis (ex.: rascunho, publicado, arquivado).
- O `status` deve ser validado e controlado pela API.

2.5 Cadastro com Dados de API Externa

- Ao cadastrar um livro, permitir informar um identificador externo (ISBN).

- Quando informado, buscar informações na API OpenLibrary Books para preencher automaticamente campos como título e número de páginas, se não forem fornecidos.

2.6 Busca e Paginação

- Endpoint para buscar materiais por título, autor ou descrição.
- Resultados paginados.

2.7 Testes

Implementar testes automatizados cobrindo:

- Validações de dados.
- Permissões de acesso.
- Funcionalidades de busca e paginação.
- Consumo da API externa.

2.8 Documentação

- Documentar a API. O README deve explicar rotas, autenticação e exemplos de uso.
- Diferencial: documentação interativa (Swagger/Rswag) ou mini frontend para interação.

2.9 GraphQL

- Diferencial: implementar ao menos uma consulta com GraphQL para acessar dados de materiais e autores.

2.10 Banco de Dados

- O projeto deve rodar em Postgres ou MySQL.

---

3. Validações Obrigatórias

3.1 Usuário

Campo	Regras
Email	Obrigatório; único; formato válido de e-mail.
Senha	Obrigatória; mínimo de 6 caracteres.

3.2 Material (todos os tipos)

Campo	Regras

Título	Obrigatório; entre 3 e 100 caracteres; não nulo/vazio.
Descrição	Opcional; quando informada, até 1000 caracteres.
Status	Obrigatório; apenas valores válidos (ex.: rascunho, publicado, arquivado).
Autor	Obrigatório; deve referenciar autor existente (pessoa ou instituição).
Usuário criador	Obrigatório; associado ao usuário que cadastrou.

3.3 Livro

Campo	Regras
ISBN	Obrigatório; único; exatamente 13 caracteres numéricos (ISBN-13).
Número de páginas	Obrigatório; maior que zero.

3.4 Artigo

Campo	Regras
DOI	Obrigatório; único; seguir o formato padrão de DOI (ex.: 10.1000/xyz123).

3.5 Vídeo

Campo	Regras
Duração (minutos)	Obrigatória; número inteiro maior que zero.

3.6 Autor (Pessoa)

Campo	Regras
Nome	Obrigatório; entre 3 e 80 caracteres.
Data de nascimento	Obrigatória; data válida; não pode ser futura.

3.7 Autor (Instituição)

Campo	Regras
Nome	Obrigatório; entre 3 e 120 caracteres.
Cidade	Obrigatória; entre 2 e 80 caracteres.

---

4. Validações Relacionais e Regras de Negócio

- Um material não pode ser excluído ou alterado por outro usuário que não seja o criador.
- Não deve ser possível cadastrar dois materiais com o mesmo identificador único (ISBN para livros, DOI para artigos).
- Não deve ser possível associar um material a um autor inexistente.
- Um autor pode ser associado a vários materiais; um material deve ter apenas um autor.
- O `status` de um material só pode ser alterado para um valor válido.
- Campos obrigatórios devem gerar erro claro e status HTTP apropriado na ausência.

---

5. Critérios de Avaliação

- Organização, clareza e padrão REST das rotas e controllers.
- Qualidade das validações e testes.
- Uso correto de associações, permissões e validações.
- Uso correto de padrões do Rails.
- Clareza e detalhamento da documentação.
- Implementação dos diferenciais.

---

6. Diferenciais

- Deploy online (Heroku, Render, Fly, Railway, etc.).

- Documentação interativa (Swagger, Postman, etc.).
- Mini frontend simples para consumir a API.
- Cobertura de testes acima de 80%.
- Endpoint GraphQL.

---

7. Entrega

- Link para o repositório (GitHub, GitLab, etc.).
- README com setup do projeto, exemplos de uso, instruções de autenticação e explicação das regras de negócio implementadas.

---

8. Dicas Gerais

- Reflita sobre como modelar materiais genéricos e específicos e como associar autores de tipos diferentes.
- Garanta as regras de acesso no código e nos testes.
- Use padrões REST, status HTTP e mensagens de erro claras.
- Estruture o projeto para facilitar manutenção e testes.

---

9. Prioridades

- API RESTful com autenticação e permissões.
- Materiais com campos genéricos/específicos e autores variados.
- Status para materiais.
- Busca e paginação.
- Consumo de API externa para cadastro.
- Testes e documentação.
- Diferenciais (quando possível).