Planejamento de teste API challenger final

Plano de Testes: API Cinema App

Objetivo do Documento

Este documento detalha a estratégia, o escopo e os casos de teste para a validação da API RESTful do sistema "Cinema App". O objetivo é garantir que todas as funcionalidades implementadas atendam aos requisitos de negócio (descritos nas Histórias de Usuário), mantenham a integridade dos dados, tratem erros de forma adequada e apliquem as regras de segurança (autenticação e autorização) corretamente.

Escopo dos Testes

Em Escopo

- **Testes de Funcionalidade:** Validação de todos os endpoints da API para as entidades:
  - Autenticação (Auth)
  - Usuários (Users)
  - Filmes (Movies)
  - Salas de Cinema (Theaters)
  - Sessões (Sessions)
  - Reservas (Reservations)
- **Testes de Integração:** Verificação da interação entre os diferentes módulos (ex: criar uma reserva deve atualizar o status dos assentos em uma sessão).
- **Testes de Validação e Tratamento de Erros:** Garantir que a API retorne códigos de status HTTP e mensagens de erro apropriadas para entradas inválidas, recursos não encontrados, etc.
- **Testes de Segurança (Autorização):** Assegurar que os endpoints protegidos só possam ser acessados por usuários autenticados e com a permissão (role) correta (`user` vs. `admin`).

Fora de Escopo

- **Testes de Interface de Usuário (Frontend):** O foco é exclusivamente na API backend.
- **Testes de Performance e Carga:** Testes de estresse para verificar o comportamento da API sob um grande volume de requisições (poderia ser uma fase futura).
- **Testes de Usabilidade:** Relacionado à experiência do usuário no frontend.

Níveis de Teste

1. **Testes de Unidade:** Foco em funções isoladas (ex: `generateToken`, métodos do Mongoose nos Models como `matchPassword`, ou um controller específico com suas dependências mockadas).
2. **Testes de Integração:** Foco na interação entre controllers, serviços e banco de dados.
3. **Testes de API (Ponta a Ponta - E2E):** Foco em simular o uso real da API através de requisições HTTP, validando o fluxo completo. *Este plano focará principalmente neste nível.*

Estratégia de Dados de Teste

- **Dados Iniciais (Seeding):** O banco de dados de teste deve ser populado antes da execução dos testes usando os scripts fornecidos (`npm run seed`). Isso garante um estado inicial conhecido.
- **Usuários de Teste:** Serão utilizados dois perfis principais:
  - **Usuário Admin:** `admin@example.com` | `password123`

    - **Usuário Comum:** `user@example.com` | `password123`
- **Dados Dinâmicos:** Para testes destrutivos (como `DELETE`) ou de criação, os dados serão criados e, se possível, removidos no próprio escopo do teste para garantir a independência e repetibilidade dos testes.

Ferramentas Recomendadas

- **Cliente de API (Testes Manuais/Exploratórios):** Postman ou Insomnia.
- **Framework de Testes Automatizados: Jest** com **Supertest** (uma combinação popular para testar APIs em Node.js).
- **CI/CD:** GitHub Actions para rodar os testes automaticamente a cada push ou pull request.

Casos de Teste Detalhados

Módulo 1: Autenticação (`/api/v1/auth`)

ID do Teste	Descrição	Passos	Resultado Esperado
AUTH-001	Registrar um novo usuário com sucesso	1. Enviar `POST` para `/register` com `name`, `email` (único) e `password` válidos.	• Status `201 Created`. - Resposta contém `success: true` e os dados do usuário com um token JWT.
AUTH-002	Tentar registrar um usuário com e-mail já existente	1. Enviar `POST` para `/register` com um e-mail que já existe no banco.	• Status `400 Bad Request`. - Mensagem de erro "User already exists".
AUTH-003	Fazer login com credenciais válidas	1. Enviar `POST` para `/login` com `email` e `password` corretos.	• Status `200 OK`. - Resposta contém `success: true` e os dados do usuário com um novo token JWT.
AUTH-004	Tentar fazer login com senha incorreta	1. Enviar `POST` para `/login` com `email` correto e `password` incorreto.	• Status `401 Unauthorized`. - Mensagem de erro "Invalid email or password".
AUTH-005	Acessar perfil (`/me`) com token válido	1. Fazer login para obter um token. 2. Enviar `GET` para `/me` com o header `Authorization: Bearer <token>`.	• Status `200 OK`. - Resposta contém os dados do usuário (sem a senha).
AUTH-006	Tentar acessar perfil (`/me`) sem token	1. Enviar `GET` para `/me` sem o header de autorização.	• Status `401 Unauthorized`. - Mensagem de erro indicando falta de autorização.
AUTH-007	Atualizar o perfil do usuário	1. Fazer login para obter um token. 2. Enviar `PUT` para `/profile` com um novo `name`, com uma nova senha.	• Status `200 OK`. - Resposta contém os dados do usuário atualizados.

AUTH-008	Atualizar o perfil do usuário	1. Fazer login para obter um token. 2. Enviar `PUT` para `/profile` apenas com um novo `name`.	• Status `200 OK`. - Resposta contém os dados do usuário atualizados.

Módulo 2: Gerenciamento de Usuários (`/api/v1/users`) (Casos de Teste Sugeridos)

Esta seção foca nas ações que um administrador pode realizar sobre todos os usuários do sistema.

ID do Teste		Descrição	Passos	Resultado Esperado
USR-001		(Admin) Listar todos os usuários	1. Fazer login como Admin. 2. Enviar GET para `/users`.	**Status 200 OK.** - A resposta contém uma lista de objetos de usuário. - **VERIFICAÇÃO CRÍTICA:** Nenhum dos objetos de usuário deve conter o campo `password`.
USR-002		(Segurança) Tentar listar usuários como usuário comum	1. Fazer login como Usuário Comum. 2. Enviar GET para `/users`.	**Status 403 Forbidden.** - Mensagem de erro indicando falta de permissão.
USR-003		(Admin) Obter um usuário específico por ID	1. Obter um `userId` da listagem de USR-001. 2. Fazer login como Admin. 3. Enviar GET para `/users/{id}`.	**Status 200 OK.** - A resposta contém os dados do usuário solicitado (sem a senha).
USR-004		(Admin) Atualizar os dados de um usuário	1. Obter um `userId`. 2. Fazer login como Admin. 3. Enviar PUT para `/users/{id}` com um `name` ou `role` atualizado.	**Status 200 OK.** - A resposta contém os dados do usuário com as informações atualizadas.
USR-005		(Segurança) Tentar atualizar outro usuário como usuário comum	1. Obter o ID do usuário Admin. 2. Fazer login como Usuário Comum. 3. Enviar PUT para `/users/{adminId}`.	**Status 403 Forbidden.**
USR-006		(Admin) Deletar um usuário	1. Criar um novo usuário para o teste. 2. Fazer login como Admin. 3. Enviar	**Status 200 OK.** - Mensagem "User removed".

			DELETE para `/users/{id_do_novo_usuario}`.	
USR-007		(Integridade) Verificar se o usuário deletado não consegue mais logar	1. Executar os passos do USR-006. 2. Tentar fazer login com as credenciais do usuário que foi deletado.	**Status 401 Unauthorized.** - Mensagem de erro "Invalid email or password".

Módulo 3: Filmes (`/api/v1/movies`)

ID do Teste	Descrição	Passos	Resultado Esperado
MOV-001	Listar todos os filmes (acesso público)	1. Enviar `GET` para `/movies`.	• Status `200 OK`. - A resposta contém uma lista de filmes.
MOV-002	Obter detalhes de um filme por ID (acesso público)	1. Enviar `GET` para `/movies/{id}` com um ID válido.	• Status `200 OK`. - A resposta contém os dados completos do filme solicitado.
MOV-003	Tentar obter um filme com ID inválido/inexistente	1. Enviar `GET` para `/movies/{id}` com um ID que não existe.	• Status `404 Not Found`. - Mensagem "Movie not found".
MOV-004	**(Admin)** Criar um novo filme	1. Fazer login como **Admin**. 2. Enviar `POST` para `/movies` com dados de um novo filme.	• Status `201 Created`. - A resposta contém os dados do filme criado.
MOV-005	**(Segurança)** Tentar criar um novo filme como **usuário comum**	1. Fazer login como **Usuário Comum**. 2. Enviar `POST` para `/movies`.	• Status `403 Forbidden`. - Mensagem de erro indicando que o usuário não tem permissão.
MOV-006	**(Admin)** Deletar um filme	1. Fazer login como **Admin**. 2. Enviar `DELETE` para `/movies/{id}` com um ID válido.	• Status `200 OK`. - Mensagem "Movie removed".
MOV-007	**(Segurança)** Tentar deletar um filme como **usuário comum**	1. Fazer login como **Usuário Comum**. 2. Enviar `DELETE` para `/movies/{id}`.	• Status `403 Forbidden`.

módulo 4: Sessões (`/api/v1/sessions`)

ID do Teste	Descrição	Passos	Resultado Esperado