Plano de Testes - API Cinema Challenge

1. Visão Geral e Objetivos

Este documento detalha o plano de testes para a API "Cinema App", desenvolvido como parte do Challenge técnico do PB AWS & AI for QE. O objetivo principal é garantir a funcionalidade, robustez e segurança dos endpoints da API, seguindo uma abordagem estruturada de testes. A validação será realizada através de testes exploratórios manuais com o Postman e, posteriormente, com a criação de uma suíte de testes automatizados com Robot Framework. Os defeitos e sugestões de melhoria encontrados serão reportados como **"Issues" diretamente no repositório do projeto no GitHub**.

2. Escopo dos Testes

2.1. Funcionalidades em Escopo

O escopo deste plano de testes abrange a validação funcional dos seguintes endpoints da API, considerando os diferentes níveis de acesso (público, usuário autenticado e administrador).

1. **Módulo de Autenticação**

- **Registro (Create):** Criação de um novo usuário.
- **Login (Read):** Autenticação de um usuário existente e geração de token de acesso.
- **Perfil do Usuário (Read):** Obtenção dos dados do usuário atualmente logado (requer token).

2. **Módulo de Usuários (Acesso Restrito a Administradores)**

- **Listagem (Read):** Obtenção da lista de todos os usuários (requer token de admin).
- **Busca por ID (Read):** Obtenção de um usuário específico pelo seu ID (requer token de admin).
- **Atualização (Update):** Modificação dos dados de um usuário (requer token de admin).
- **Exclusão (Delete):** Remoção de um usuário do sistema (requer token de admin).

3. **Módulo de Filmes**

- **Listagem (Read):** Obtenção da lista de todos os filmes (público).
- **Busca por ID (Read):** Obtenção dos detalhes de um filme específico (público).
- **Criação (Create):** Adição de um novo filme ao catálogo (requer token de admin).
- **Atualização (Update):** Modificação dos dados de um filme (requer token de admin).
- **Exclusão (Delete):** Remoção de um filme do catálogo (requer token de admin).

4. **Módulo de Teatros (Salas)**

- **Listagem (Read):** Obtenção da lista de todos os teatros/salas (público).
- **Busca por ID (Read):** Obtenção dos detalhes de um teatro específico (público).
- **Criação (Create):** Adição de um novo teatro (requer token de admin).
- **Atualização (Update):** Modificação dos dados de um teatro (requer token de admin).
- **Exclusão (Delete):** Remoção de um teatro (requer token de admin).

5. **Módulo de Sessões**

- **Listagem (Read):** Obtenção da lista de todas as sessões disponíveis (público).
- **Busca por ID (Read):** Obtenção dos detalhes de uma sessão específica (público).
- **Criação (Create):** Agendamento de uma nova sessão (requer token de admin).
- **Atualização (Update):** Modificação de uma sessão existente (requer token de admin).
- **Exclusão (Delete):** Cancelamento de uma sessão (requer token de admin).

6. **Módulo de Reservas**

- **Criação (Create):** Realização de uma nova reserva por um usuário autenticado.
- **Listagem de Minhas Reservas (Read):** Consulta das reservas do usuário atualmente logado.
- **Busca por ID (Read):** Obtenção dos detalhes de uma reserva específica (acesso de admin ou do dono da reserva).
- **Listagem Geral (Read):** Obtenção da lista de todas as reservas do sistema (requer token de admin).
- **Atualização de Status (Update):** Modificação do status de uma reserva (requer token de admin).
- **Exclusão (Delete):** Remoção de uma reserva (requer token de admin).

3.Ferramentas e Ambiente

- **Postman:** testes exploratórios
- **Xmind:** produção do mapa mental

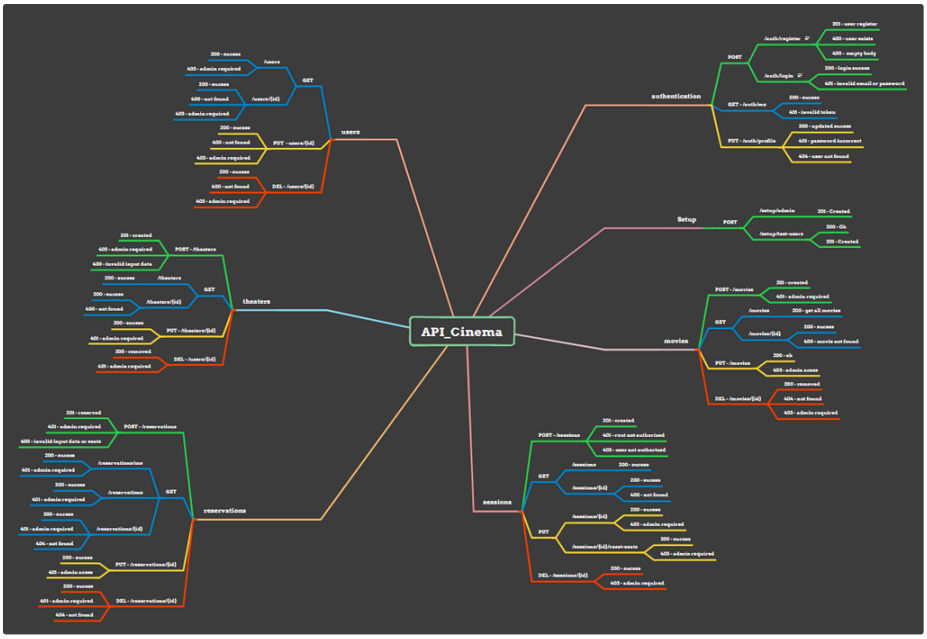
- **Robot Framework:** para automaçao
- **GitHub:** para versionamento e gestão de issue
- **Confluence:** para relatorios
- **Jira:** para documentação e gráficos via QAlity

4. Estratégia de Teste

A estratégia adotada é a de **Testes Exploratórios Guiados por Cenários**. Inicialmente, cada funcionalidade é explorada manualmente com o **Postman** para entender seu comportamento e identificar cenários de sucesso e falha. Essas descobertas são então formalizadas em um mapa mental de cobertura no **Xmind** e documentadas nesta página. A fase final consistirá na automação dos principais cenários utilizando **Robot Framework**.

5. Cobertura de Teste e Cenários

5.1 Mapa Mental de Cobertura



5.2 Cenários de Teste Detalhados

Funcionalidade: Autenticação

ID	TITULO	PASSOS	RESULTADO ESPERADO
CT-AU-01	Registro de usuário com sucesso	1. Enviar requisição `POST` para `/auth/register` com dados válidos e um e-mail inédito.	Status `201 Created`. Corpo da resposta contém `success:true` e um token JWT.
CT-AU-02	Tentar registrar com e-mail duplicado	1. Enviar requisição `POST` para `/auth/register` com um e-mail que já existe no sistema.	Status `400 Bad Request`. Corpo da resposta contém `success:false` e a mensagem "User already exists".
CT-AU-03	Tentar registrar com corpo vazio	1. Enviar requisição `POST` para `/auth/register` com o corpo `{}`.	Status `400 Bad Request`. Corpo da resposta contém um objeto `errors` detalhando os campos faltantes.
CT-AU-04	Login de usuário com sucesso	1. Enviar requisição `POST` para `/auth/login` com email e senha válidos.	Status `200 OK`. Corpo da resposta contém `success:true` e um token JWT.
CT-AU-05	Tentar login com credenciais inválidas	1. Enviar requisição `POST` para `/auth/login` com uma senha incorreta.	Status `401 Unauthorized`.
CT-AU-06	Obter perfil do usuário com	1. Fazer login e obter um token válido.	Status `200 OK`. A resposta contém os dados do perfil do

	sucesso (token válido)	2. Enviar requisição `GET` para `/auth/me` com o token no cabeçalho `Authorization`.	usuário.
CT-AU-07	Tentar obter perfil com token inválido/ausente	1. Enviar requisição `GET` para `/auth/me` SEM um token válido no cabeçalho `Authorization`.	Status `401 Unauthorized` com mensagem de erro apropriada.
CT-AU-08	Validar atualização de perfil com senha atual	1. Fazer login e obter um token válido. 2. Enviar requisição `PUT` para `/auth/profile` com a palavra-passe atual correta e novos dados de perfil.	Status `200 OK` com os dados do perfil atualizados.
CT-AU-09	Login de admin com sucesso	1. Enviar requisição `POST` para `/auth/login` com email e senha válidos.	Status `200 OK`. Corpo da resposta contém `success:true` e um token JWT.

Funcionalidade: Setup

ID	TITULO	PASSOS	RESULTADO ESPERADO
CT-SET-01	Criar usuário `Admin` individualmente	1. Enviar uma requisição `POST` para o endpoint `Create admin user`. 2. No corpo (`Body`), enviar os dados de um novo usuário com o perfil (`role`) definido como `Admin`.	**Status:** `201 Created`. O corpo da resposta deve indicar sucesso e conter os dados do usuário `Admin` criado.
CT-SET-02	Popular banco com usuários de teste	1. Garantir que os usuários de teste (`test@example.com`, etc.) não existam no banco de dados. 2. 2. Enviar uma requisição `POST` para o endpoint `/setup/test-users`.	**Status:** `201 Created`. O corpo da resposta deve listar os novos usuários no array `"created"`,
CT-SET-03	Validar criação de usuários de teste já existentes	1. Garantir que os usuários de teste **já existam** no banco (executar CT-SE-02 antes). 2. Enviar uma requisição `POST` para o endpoint `/setup/test-users` novamente.	**Status:** `201 OK`. e listar os usuários no array

Funcionalidade: Filmes

ID	TITULO	PASSOS	RESULTADO ESPERADO
CT-FI-01	Listar todos os titulos de filmes	1. Fazer uma requisição `GET` para o endpoint `/movies`.	A API deve retornar o **código de status** `200 OK` e o corpo da resposta deve conter um JSON com uma lista de filmes dentro da chave `"data"`.
CT-FI-02	Localizar titulo depor `{id}` válido	1. Obter um `movie_id` válido de um filme existente (ex: através da resposta do CT-FI-01). 2. Fazer uma requisição `GET` para o endpoint `/movies/{movie_id}` substituindo `{movie_id}` pelo ID obtido.	A API deve retornar o **código de status** `200 OK` e o corpo da resposta deve conter um objeto JSON com os dados completos do filme solicitado.
CT-FI-03	Localizar titulo por `{id}` inválido	1. Fazer uma requisição `GET` para o endpoint `/movies/{id}` utilizando um ID que sabidamente não existe (ex: `1234567890abc`).	A API deve retornar o **código de status** `404 Not Found` com mensagem de erro apropriada.
CT-FI-04	Cadastrar novo filme em cartaz	1. Obter um token de autenticação de um usuário com perfil de **Admin**.	A API deve retornar o código de status `201 Created`.

		2. Fazer uma requisição do tipo `POST` para o endpoint `/movies`. 3. Na aba `Authorization`, configurar `Bearer Token` com o token admin. 4. Na aba `Body`, enviar um JSON com dados válidos para a criação de um filme (ex: `title`, `genres`, `director`, `duration`, `poster`).	O corpo da resposta deve conter um objeto JSON com todos os dados do filme recém-criado, incluindo um `id` único gerado pelo sistema.
CT-FI-05	Tentar cadastrar um novo filme sem permissão de admin (Negative Path)	1. Obter um token de autenticação de um usuário **comum** (não-administrador). 2. Fazer uma requisição do tipo `POST` para o endpoint `/movies`. 3. Na aba `Authorization`, configurar o `Bearer Token` com o token de usuário comum. 4. Na aba `Body`, enviar um JSON com dados válidos de um filme.	A API deve retornar o código de status `401 Unauthorized` (ou `403 Forbidden`, conforme a implementação da API). com mensagem de erro apropriada.
CT-FI-06	Atualizar dados de um filme existente com sucesso	1. Executar os passos do caso de teste `CT-FI-04` para garantir que um filme exista e para obter seu `id`. 2. Fazer uma requisição do tipo `PUT` para o endpoint `/movies/{id}`, substituindo `{id}` pelo `id` do filme criado no passo 1. 3. Na aba `Authorization`, configurar o `Bearer Token` com o token de administrador. 4. Na aba `Body`, enviar um JSON com os dados que serão atualizados (ex: um novo `title` ou `synopsis`).	A API deve retornar o código de status `200 OK`. O corpo da resposta deve conter o objeto JSON completo do filme, com os campos refletindo as alterações realizadas.
CT-FI-07	Tentar atualizar um filme que não está cadastrado	1. Fazer uma requisição do tipo `PUT` para o endpoint `/movies/{id}`, substituindo `{id}` por `id` invalido 2. Na aba `Authorization`, configurar o `Bearer Token` com o token de administrador. 3. Na aba `Body`, enviar um JSON com os dados validos	A API deve retornar o código de status `400 resource not found`. O corpo da resposta deve conter o objeto JSON com o erro apropriado.

CT-FI-08	Tentar atalizar dados de um filme sem permissão	1. Executar os passos do caso de teste `CT-FI-03` para garantir que um filme exista e para obter seu `id`. 2. Fazer uma requisição do tipo `PUT` para o endpoint `/movies/{id}`, substituindo `{id}` pelo `id` do filme criado no passo 1. 3. Na aba `Authorization`, configurar o `Bearer Token` com o token de usuario. 4. Na aba `Body`, enviar um JSON com os dados que serão atualizados (ex: um novo `title` ou `synopsis`).	A API deve retornar o código de status `403 User role user is not authorized to access this route`.
CT-FI-09	Excluir um filme existente com sucesso	1. Executar os passos do caso de teste `CT-FI-03` para garantir que um filme exista e para obter seu `id`. 2. Fazer uma requisição do tipo `DELETE` para o endpoint `/movies/{id}`, substituindo `{id}` pelo `id` do filme criado no passo 1. 3. Na aba `Authorization`, configurar o `Bearer Token` com o token de administrador.	A API deve retornar o código de status `200 OK`. O corpo da resposta deve conter uma mensagem confirmando que o filme foi removido com sucesso.
CT-FI-09	Tentar excluir um filme sem acesso autorizado	1. Executar os passos do caso de teste `CT-FI-03` para garantir que um filme exista e para obter seu `id`. 2. Fazer uma requisição do tipo `DELETE` para o endpoint `/movies/{id}`, substituindo `{id}` pelo `id` do filme criado no passo 1. 3. Na aba `Authorization`, configurar o `Bearer Token` com o token de usuário.	A API deve retornar o código de status `403 User role user is not authorized to access this route`.
CT-FI-10	Tentar excluir um filme não existente	1. Fazer uma requisição do tipo `DELETE` para o endpoint `/movies/{id}`, substituindo `{id}` por um `id` inexistente. 2. Na aba `Authorization`, configurar o `Bearer Token` com o token de administrador.	A API deve retornar o código de status `404 resource not found`. O corpo da resposta deve conter o objeto JSON com o erro apropriado.

Funcionalidade: Sessões

ID	TITULO	PASSOS	RESULTADO ESPERADO
CT-SE-01	Listar todas as sessões com sucesso	1. Enviar uma requisição do tipo `GET` para o endpoint `/sessions`.	**Status:** `200 OK` O corpo da resposta deve conter a estrutura de paginação e uma lista (`data`)
CT-SE-02	Criar uma nova sessão como `admin`	1. Obter um token de autenticação de um usuário com perfil de **Admin**.	**Status:** `201 created.` `"success": true` . Um objeto `"data"` com os detalhes da sessão criada, **incluindo um** `_id`

		2. Fazer uma requisição do tipo `POST` para o endpoint `/sessions`. 3. Na aba `Authorization`, configurar `Bearer Token` com o token admin. 4. Na aba `Body`, enviar um JSON com dados válidos para a criação de uma sessão (ex: `movie`, `theater`).	**único** gerado pelo sistema e o array `seats` correspondente à sala.
CT-SE-03	Tentar criar uma sessão com usuário não autorizado	1. Obter um token de autenticação de um usuário com perfil de **Usuario**. 2. Fazer uma requisição do tipo `POST` para o endpoint `/sessions`. 3. Na aba `Authorization`, configurar `Bearer Token` com o token invalido. 4. Na aba `Body`, enviar um JSON com dados válidos para a criação de uma sessão (ex: `movie`, `theater`).	**Status:** `403 Forbidden` `sucess:false` com detalhes de acesso negado para a rota
CT-SE-04	Obter sessão por ID com sucesso	1. **Pré-condição:** Obter um `id` de sessão válido (usar o do CT-SE-02). 2. Enviar requisição `GET` para `/sessions/{session_id}`.	**Status:** `200 OK`. **Corpo:** Objeto JSON com os dados completos da sessão.
CT-SE-05	Atualizar uma sessão existente	1. **Pré-condição:** Obter um `id` de sessão válido. 2. Obter token de um usuário `Admin`. 3. Enviar requisição `PUT` para `/sessions/{Ssession_id}` com o token e um `body` com os dados a serem atualizados.	**Status:** `200 OK`. **Corpo:** Objeto JSON com os dados da sessão refletindo as alterações.
CT-SE-06	Excluir uma sessão com sucesso	1. **Pré-condição:** Obter um `id` de sessão válido. 2. Obter token de um usuário `Admin`. 3. Enviar requisição `DELETE` para `/sessions/{session_id}` com o token de Admin.	**Status:** `200 OK`. **Corpo:** Mensagem confirmando a exclusão ou corpo vazio.
CT-SE-07	Tentar excluir sessão sem permissão	1. **Pré-condição:** Obter um `id` de sessão válido. 2. Obter token de um usuário **comum**. 3. Enviar requisição `DELETE` para `/sessions/{id_da_sessao}` com o token de usuário comum.	**Status:** `403 Forbidden`. **Corpo:** Mensagem de erro sobre falta de permissão.
CT-SE-08	Resetar assentos de uma sessão	1. **Pré-condição:** Obter um `id` de sessão válido. 2. Obter token de um usuário `Admin`. 3. Enviar requisição `PUT` para `/sessions/{session_new_id}/reset-seats` com o token.	**Status:** `200 OK`. **Corpo:** Objeto JSON onde **todos** os assentos da sessão têm o `"status": "available"`.

Funcionalidade: Teatros

ID	TITULO	PASSOS	RESULTADO ESPERADO
CT-TE-01	Listar todas as salas de cinema	1. Enviar requisição `GET` para `/theaters`.	**Status:** `200 OK`. **Corpo:** Lista com todas as salas de cinema cadastradas.

CT-TE-02	Criar uma nova sala com sucesso	1. Obter token de `Admin`. 2. Enviar requisição `POST` para `/theaters` com o token e um `body` com dados válidos (`name`, `capacity`, `type`).	**Status:** `201 Created`. **Corpo:** Objeto JSON com os dados da sala criada, incluindo um `id` único.
CT-TE-03	Tentar criar sala com dados inválidos	1. Obter token de `Admin`. 2. Enviar requisição `POST` para `/theaters` com o token e um `body` com dados inválidos (ex: `capacity` como texto).	**Status:** `400 Bad Request`. **Corpo:** Mensagem de erro informando sobre o campo inválido.
CT-TE-04	Tentar criar sala sem permissão	1. Obter token de usuário **comum**. 2. Enviar requisição `POST` para `/theaters` com o token de usuário comum.	**Status:** `403 Forbidden`. **Corpo:** Mensagem de erro sobre falta de permissão.
CT-TE-05	Obter uma sala por ID com sucesso	1. **Pré-condição:** Obter um `id` de sala válido. 2. Enviar requisição `GET` para `/theaters/{id_da_sala}`.	**Status:** `200 OK`. **Corpo:** Objeto JSON com os dados completos da sala especificada.
CT-TE-06	Atualizar uma sala existente	1. **Pré-condição:** Obter um `id` de sala válido. 2. Obter token de `Admin`. 3. Enviar requisição `PUT` para `/theaters/{id_da_sala}` com o token e um `body` com os dados a serem atualizados.	**Status:** `200 OK`. **Corpo:** Objeto JSON com os dados da sala refletindo as alterações.
CT-TE-07	Tentar atualizar sala sem permissão	1. **Pré-condição:** Obter um `id` de sala válido. 2. Obter token de usuário **comum**. 3. Enviar requisição `PUT` para `/theaters/{id_da_sala}`.	**Status:** `401 Unauthorized` ou `403 Forbidden`. **Corpo:** Mensagem de erro sobre falta de permissão.
CT-TE-08	Excluir uma sala com sucesso	1. **Pré-condição:** Obter um `id` de sala válido. 2. Obter token de `Admin`. 3. Enviar requisição `DELETE` para `/theaters/{id_da_sala}` com o token.	**Status:** `200 OK` ou `204 No Content`. **Corpo:** Mensagem confirmando a exclusão ou corpo vazio.
CT-TE-09	Tentar excluir sala sem permissão	1. **Pré-condição:** Obter um `id` de sala válido. 2. Obter token de usuário **comum**. 3. Enviar requisição `DELETE` para `/theaters/{id_da_sala}`.	**Status:** `401 Unauthorized` ou `403 Forbidden`. **Corpo:** Mensagem de erro sobre falta de permissão.

Funcionalidade: Reservas

ID	TITULO	PASSOS	RESULTADO ESPERADO
CT-RE-01	Listar todas as reservas do usuário	1. Obter token de um usuário **comum**. 2. Enviar requisição `GET` para `/reservations/me`.	**Status:** `200 OK`. **Corpo:** Lista com todas as reservas feitas pelo usuário autenticado.
CT-RE-02	Tentar listar reservas sem autenticação	1. Enviar requisição `GET` para `/reservations/` **sem** token de autorização.	**Status:** `401 Unauthorized`. **Corpo:** Mensagem de erro sobre falta de permissão/autenticação.

CT-RE-03	Criar uma nova reserva com sucesso	1. **Pré-condição:** Obter `id` de sessão e assentos (`seats`) disponíveis. 2. Obter token de usuário **comum**. 3. Enviar requisição `POST` para `/reservations` com o token e um `body` com `sessionId` e `seats`.	**Status:** `201 Created`. **Corpo:** Objeto JSON com os dados da reserva criada, incluindo um `id` único.
CT-RE-04	Tentar criar reserva sem autenticação	1. **Pré-condição:** Obter `id` de sessão e assentos. 2. Enviar requisição `POST` para `/reservations` **sem** token.	**Status:** `401 Unauthorized`. **Corpo:** Mensagem de erro sobre falta de permissão/autenticação.
CT-RE-05	Tentar criar reserva com dados inválidos	1. Obter token de usuário **comum**. 2. Enviar requisição `POST` para `/reservations` com um `body` inválido (ex: `sessionId` inexistente ou assentos já ocupados).	**Status:** `400 Bad Request`. **Corpo:** Mensagem de erro informando sobre o dado inválido.
CT-RE-06	Listar todas as reservas (Admin Acess)	1. Obter token de `Admin`. 2. Enviar requisição `GET` para `/reservations`	**Status:** `200 OK`. **Corpo:** Lista com **todas** as reservas de **todos** os usuários.
CT-RE-07	Tentar listar todas as reservas (Usuário Comum)	1. Obter token de usuário **comum**. 2. Enviar requisição `GET` para `/reservations/all`.	**Status:** `401 Unauthorized` ou `403 Forbidden`. **Corpo:** Mensagem de erro sobre falta de permissão.
CT-RE-08	Obter uma reserva por ID com sucesso	1. **Pré-condição:** Obter um `id` de reserva válido. 2. Obter token de usuário (dono da reserva) ou `Admin`. 3. Enviar requisição `GET` para `/reservations/{id_da_reserva}`.	**Status:** `200 OK`. **Corpo:** Objeto JSON com os dados completos da reserva.
CT-RE-09	Tentar obter reserva com ID inexistente	1. Obter token de `Admin` ou usuário comum. 2. Enviar requisição `GET` para `/reservations/{id_inexistente}`.	**Status:** `404 Not Found`. **Corpo:** Mensagem de erro indicando que a reserva não foi encontrada.
CT-RE-10	Atualizar status de uma reserva (Admin acess)	1. **Pré-condição:** Obter um `id` de reserva válido. 2. Obter token de `Admin`. 3. Enviar requisição `PUT` para `/reservations/{id_da_reserva}` com um `body` contendo o novo status.	**Status:** `200 OK`. **Corpo:** Objeto JSON com os dados da reserva refletindo a alteração de status.
CT-RE-11	Tentar atualizar status sem permissão	1. **Pré-condição:** Obter um `id` de reserva válido. 2. Obter token de usuário **comum**. 3. Enviar requisição `PUT` para `/reservations/{id_da_reserva}`.	**Status:** `401 Unauthorized` ou `403 Forbidden`. **Corpo:** Mensagem de erro sobre falta de permissão.
CT-RE-12	Excluir uma reserva com sucesso (Admin acess)	1. **Pré-condição:** Obter um `id` de reserva válido. 2. Obter token de `Admin`. 3. Enviar requisição `DELETE` para `/reservations/{id_da_reserva}`.	**Status:** `200 OK` ou `204 No Content`. **Corpo:** Mensagem confirmando a exclusão ou corpo vazio.

Funcionalidade: Usuarios

ID	TITULO	PASSOS	RESULTADO ESPERADO

CT-US-01	Listar todos os usuários (Admin)	1. Obter token de `Admin`. 2. Enviar requisição `GET` para `/users`.	**Status:** `200 OK`. **Corpo:** Lista com todos os usuários do sistema.
CT-US-02	Tentar listar usuários sem permissão	1. Obter token de usuário **comum**. 2. Enviar requisição `GET` para `/users`.	**Status:** `403 Forbidden`. **Corpo:** Mensagem de erro sobre falta de permissão.
CT-US-03	Obter um usuário por ID (Admin)	1. **Pré-condição:** Obter um `id` de usuário válido. 2. Obter token de `Admin`. 3. Enviar requisição `GET` para `/users/{id_do_usuario}`.	**Status:** `200 OK`. **Corpo:** Objeto JSON com os dados completos do usuário.
CT-US-04	Tentar obter usuário por ID sem permissão	1. **Pré-condição:** Obter um `id` de usuário válido. 2. Obter token de usuário **comum**. 3. Enviar requisição `GET` para `/users/{id_do_usuario}`.	**Status:** `403 Forbidden`. **Corpo:** Mensagem de erro sobre falta de permissão.
CT-US-05	Atualizar um usuário (Admin)	1. **Pré-condição:** Obter um `id` de usuário válido. 2. Obter token de `Admin`. 3. Enviar requisição `PUT` para `/users/{id_do_usuario}` com um `body` com os dados a serem atualizados.	**Status:** `200 OK`. **Corpo:** Objeto JSON com os dados do usuário refletindo as alterações.
CT-US-06	Tentar atualizar usuário sem permissão	1. **Pré-condição:** Obter um `id` de usuário válido. 2. Obter token de usuário **comum**. 3. Enviar requisição `PUT` para `/users/{id_do_usuario}`.	**Status:** `403 Forbidden`. **Corpo:** Mensagem de erro sobre falta de permissão.
CT-US-07	Excluir um usuário (Admin acess)	1. **Pré-condição:** Obter um `id` de usuário válido (diferente do admin logado). 2. Obter token de `Admin`. 3. Enviar requisição `DELETE` para `/users/{id_do_usuario}`.	**Status:** `200 OK` **Corpo:** Mensagem confirmando a exclusão.
CT-US-08	Tentar excluir usuário sem permissão	1. **Pré-condição:** Obter um `id` de usuário válido. 2. Obter token de usuário **comum**. 3. Enviar requisição `DELETE` para `/users/{id_do_usuario}`.	**Status:** `403 Forbidden`. **Corpo:** Mensagem de erro sobre falta de permissão.

5. Gestão de Defeitos (Issues)

falhas ou sugestões de melhoria encontrados serão registrados e gerenciados como "Issues" do tipo "Bug" no **Jira**. A gestão dos casos de teste e a criação de dashboards de acompanhamento serão feitas com o app **QAlity for Jira / GitHub Issues**.

Cada bug reportado deve conter um título descritivo, os passos para reproduzir o erro, o resultado atual e o resultado esperado, além de evidências (screenshots, logs).

Link para o Projeto no Jira: Cinema App Challenge | Quadro

6. Testes de Fluxo (End-to-End)

ID	TITULO	PASSOS	RESULTADO ESPERADO
CT-E2E-01	Ciclo de Vida de uma Reserva	1. **(Admin)** Fazer login. 2. **(Admin)** Criar um novo Filme. 3. **(Admin)** Criar uma nova Sala (Theater). 4. **(Admin)** Criar uma nova Sessão para o filme e sala criados. 5. **(Usuário Comum)** Fazer login.	**Status:** Todas as etapas retornam o status de sucesso esperado (200, 201). **Corpo:** A reserva final é criada corretamente com

		6. **(Usuário Comum)** Criar uma Reserva para 2 assentos na sessão. 7. **(Usuário Comum)** Listar "minhas reservas" e validar que a nova reserva aparece.	os dados consistentes das etapas anteriores.
CT-E2E-02	Ciclo de Gestão de um Filme (Admin)	1. **(Admin)** Fazer login. 2. **(Admin)** Criar um novo Filme e guardar o `id`. 3. **(Admin)** Buscar o filme pelo `id` e validar que os dados estão corretos. 4. **(Admin)** Atualizar o filme (ex: mudar o `title`). 5. **(Admin)** Buscar o filme pelo `id` novamente e validar que a alteração foi aplicada. 6. **(Admin)** Excluir o filme. 7. **(Admin)** Tentar buscar o filme pelo `id` uma última vez.	**Status:** Passos 1-6 retornam sucesso (200/201). Passo 7 retorna `404 Not Found`. **Validação:** O fluxo completo de CRUD (Create, Read, Update, Delete) para um recurso principal funciona como esperado.
CT-E2E-03	Ciclo de Vida do Usuário (Do Registro à Exclusão)	1. **(Novo Usuário)** Registrar um novo usuário via `POST /auth/register`. Guardar o `id` e `email`. 2. **(Novo Usuário)** Fazer login com as credenciais recém-criadas para obter um token. 3. **(Novo Usuário)** Buscar o próprio perfil (`GET /auth/me`) e validar se os dados estão corretos. 4.**(Novo Usuário)** Atualizar o próprio perfil (`PUT /auth/profile`), por exemplo, mudando o nome. 5. **(Admin)** Fazer login como `Admin` para obter um token de administrador. 6. **(Admin)** Usar o token de Admin para excluir a conta do usuário criada no passo 1 (`DELETE /users/{id_do_novo_usuario}`). 7. **(Validação Final)** Tentar fazer login novamente com as credenciais do usuário excluído.	**Status:** Passos 1-6 retornam sucesso (200/201). **Validação Final:** O passo 7 deve falhar com status `401 Unauthorized`, provando que o usuário foi removido com sucesso.