Plano de Testes — Cinema App

Semana 15 & 16 - Challenge Final
Plano de Testes
Versão 1.0.3

@Pedro Afonso de Alencar Silva

📜 Histórico de Revisões

Versão	Data	Descrição
1.0	24 de jun. de 2025	• Versão Inicial
1.0.1	25 de jun. de 2025	• + 12 CTs Adicionar em "10. Matriz de Cenários de Teste e Priorização" - Pós-Testes Exploratórios e Pós-Reanálise das US. • + Preenchimento da Seção "12. Testes Candidatos à Automação".
1.0.2	26 de jun. de 2025	• - CT038 da "10. Matriz de Cenários de Teste e Priorização"
1.0.3	26 de jun. de 2025	• + Comentários Informando Correções na Documentação do Swagger em "5.3. Mapa Mental da API" • + Novo Mapa Mental Pós Correções na Documentação do Swagger em "5.3. Mapa Mental da API"
1.0.3	2 de jul. de 2025	• - CT029 de "12.1. Candidatos para

		Automação de API (Back-end)”

📑 Índice

1. 📖 Glossário

Termo	Significado
API	(Application Programming Interface) Conjunto de regras e ferramentas para construir software. No projeto, é o back-end que gerencia os dados do cinema.
Back-end	A parte "invisível" do sistema, o "cérebro" da aplicação, onde a lógica de negócio, as regras e o acesso ao banco de dados acontecem.
Front-end	A parte do sistema com a qual o usuário interage diretamente. A interface gráfica (UI) no navegador.
Endpoint	Uma URL específica na API onde uma funcionalidade pode ser acessada (ex: `/api/v1/login`).
Heurísticas de Teste	"Regras de bolso" ou atalhos mentais usados para guiar o teste exploratório e encontrar problemas comuns de forma eficiente.
CI/CD	(Integração Contínua / Entrega Contínua) Prática de automação para integrar mudanças de código e executar testes automaticamente.
PageObjects	Padrão de projeto em automação de UI onde cada página é mapeada como um objeto, centralizando os localizadores para facilitar a manutenção.
ServiceObjects	Análogo ao PageObjects, mas para testes de API. Agrupa as chamadas de API relacionadas a um serviço (ex: serviço de "Filmes") em um arquivo.
Caminho Feliz	O cenário de teste ideal, sem erros ou exceções, onde o usuário segue o fluxo principal da funcionalidade com sucesso.

Cenário Negativo	Um cenário que verifica como o sistema lida com entradas inválidas, erros do usuário ou condições inesperadas.
Teste de Regressão	Testes realizados para garantir que as novas alterações no código não quebraram funcionalidades que já existiam.
Teste Exploratório	Abordagem de teste "livre", sem roteiro, onde o testador aprende sobre a aplicação e busca por bugs de forma dinâmica.
User Story (US)	Descrição curta de uma funcionalidade sob a perspectiva do usuário. É a base para o desenvolvimento e para a criação de cenários de teste.
Robot Framework	O framework de automação de testes utilizado no projeto, conhecido por sua sintaxe baseada em keywords.
Keyword	No Robot Framework, uma ação ou passo de teste definido em linguagem natural (ex: "Abrir Navegador", "Clicar Botão").
Swagger	Ferramenta para projetar, construir e documentar APIs. No projeto, é a documentação interativa da API que você analisou.
Token (Bearer Token)	Chave de acesso gerada após o login, usada nas requisições para provar que o usuário está autenticado e autorizado.
Request Body	A carga de dados (geralmente em JSON) enviada com requisições `POST` ou `PUT` para criar ou atualizar um recurso na API.
Schema	A definição da estrutura de dados esperada pela API, tanto para o que é enviado (Request) quanto para o que é recebido (Response).
Pirâmide de Testes	Metodologia que distribui o esforço de automação em diferentes camadas (Unidade, Serviço/API, UI), priorizando testes mais rápidos e estáveis na base.

Postman	Ferramenta usada para fazer requisições HTTP, testar e documentar APIs. No projeto, é o "laboratório" para a exploração manual do back-end.
GitHub Actions	Plataforma de automação integrada ao GitHub que permite criar fluxos de trabalho (workflows) de CI/CD, como executar testes automaticamente.
Google Gemini API	API Gemini possibilita a utilização de modelos generativos mais recentes do Google. No projeto, utilizado para validação e formatação de alguns responses.

2. 📘 Apresentação

O Cinema App é uma aplicação web voltada para reserva de ingressos de cinema e gestão de sessões, filmes e salas. A solução é composta por:

- **Back-end:** API RESTful desenvolvida com Node.js, Express e MongoDB, responsável por autenticação, gerenciamento de entidades (filmes, salas, sessões) e controle de reservas.
- **Front-end:** Aplicação desenvolvida com React 18 e Vite, focada em uma experiência de usuário fluida, com recursos como seleção de assentos, histórico de reservas e painel administrativo.

A aplicação atende tanto **usuários comuns quanto administradores**, com funcionalidades adaptadas para cada perfil.

3. 🎯 Objetivo

Este plano de testes visa garantir que o Front-end e o Back-end do Cinema App estejam conforme as **regras de negócio definidas nas User Stories**, além de **validar a consistência e completude** da documentação técnica (Swagger), **reportando as divergências encontradas**.

A proposta é validar tanto cenários principais quanto alternativos, incluindo fluxos não explicitamente documentados, a fim de assegurar a confiabilidade, rastreabilidade e qualidade da aplicação.

Além disso, este plano busca:

- Identificar **cenários candidatos à automação**;
- Apoiar uma **cobertura de testes orientada a risco**;
- Gerar **evidências claras de execução** com base nos critérios de aceite definidos.

4. 📌 Escopo

4.1. Dentro do Escopo

Este plano contempla a **validação funcional e de regras de negócio** descritas nas **User Stories** disponíveis em:

Type	Resumo	Prioridade	Status
✨	Histórias de Experiência do Usuário	= Medium	A FAZER

Type	Resumo	Prioridade	Status
	Histórias de Reserva	= Medium	A FAZER
	Histórias de Gerenciamento de Sessões	= Medium	A FAZER
	Histórias de Gerenciamento de Filmes	= Medium	A FAZER
	Histórias de Autenticação	= Medium	A FAZER
	US-NAV-001: Navegação Intuitiva	= Medium	A FAZER
	US-RESERVE-003: Visualizar Minhas Reser...	= Medium	A FAZER
	US-RESERVE-002: Processo de Checkout	= Medium	A FAZER
	US-RESERVE-001: Selecionar Assentos par...	= Medium	A FAZER
	US-SESSION-001: Visualizar Horários de S...	= Medium	A FAZER
	US-MOVIE-002: Visualizar Detalhes do Filme	= Medium	A FAZER

Sincronizado agora • 17 itens

1. **Testes Funcionais Adicionais:**

- Validação ponta a ponta dos **fluxos críticos de reserva e gerenciamento**;
- Execução de **testes de regressão** com foco em impactos de alterações;

2. **Testes Não-Funcionais:**

- **Teste de Usabilidade**: verificação de navegação intuitiva, clareza das mensagens de erro e consistência visual;
- **Validação da Documentação Swagger**: comparação entre o comportamento real da API e sua documentação, reportando desvios ou omissões;
- **Tratamento de Erros da API**: validação de respostas frente a dados inválidos, requisições malformadas e estados inesperados.

3. **Atividades e Entregáveis do Processo de Teste:**

- Execução de testes exploratórios;
- Abertura e gerenciamento de bugs com evidências claras;

4.2. Fora do Escopo

- **Testes de Performance, Carga e Estresse**: não serão realizados testes relacionados à escalabilidade, tempo de resposta sob alta demanda ou resistência da aplicação.
- **Testes de Segurança Aprofundados**: validações básicas de autenticação e autorização serão realizadas, mas **testes de vulnerabilidades** (como SQL Injection, XSS, etc.) não estão incluídos.
- **Testes de Compatibilidade Extensivos**: os testes se limitarão a **3 navegadores principais** (Chromium, Firefox e WebKit), não abrangendo múltiplas versões ou navegadores menos comuns.
- Durante a fase inicial de testes exploratórios, foi identificado que o **painel de administração do front-end retorna erro 404 (Página Não Encontrada)**, impossibilitando a validação completa das funcionalidades administrativas. Este achado será reportado como uma **issue no repositório** para análise e correção.
- **Funcionalidades Inacabadas:** O endpoint de atualização de senha (`PUT /auth/profile`) está exposto na API, mas não funcional, levando a erros internos e falsos positivos. O front-end alerta que a função de alterar

senha estará disponível em breve. Por enquanto, apenas a atualização do nome é suportada.

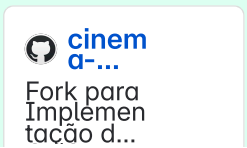
5. 🔍 Análise

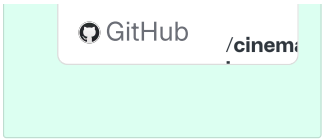
Esta seção detalha os riscos, dependências e premissas identificados durante a fase de exploração do Cinema App, além de delinear as estratégias de teste adotadas para mitigar esses riscos.

5.1. Análise Técnica e de Implementação

Esta seção foca nos riscos identificados através da exploração técnica da aplicação e da sua documentação.

Risco Identificado	Impacto no Teste	Estratégia de Mitigação / Contingência
Documentação da API Incompleta: Endpoints de `/users` e outros estão ausentes ou mal documentados no Swagger.	Dificulta a descoberta e o uso correto das funcionalidades da API, tornando o Swagger uma fonte não confiável.	A exploração manual via **Postman será tratada como a "fonte da verdade"**. A automação da API será baseada no comportamento observado, e as divergências serão reportadas como bugs de documentação. ✅ Correção Pós-Plano de Testes: 
Funcionalidades Administrativas Inacessíveis no Front-end: A interface de administração para cadastrar filmes, salas e sessões não está implementada.	Impede a execução de testes de ponta a ponta para os fluxos administrativos através da interface gráfica (UI).	As funcionalidades administrativas serão testadas **diretamente na camada de API**. Os dados necessários para os testes de UI (filmes, sessões) serão criados via requisições de API no `Setup` dos testes automatizados.
Funcionalidades Inacabadas: O endpoint de atualização de senha (`PUT /auth/profile`)	Pode levar à criação de testes automatizados para uma funcionalidade quebrada, desperdiçando	O teste exploratório confirmou a inconsistência. Esta funcionalidade será **marcada como fora do**

está exposto na API, mas não funcional, levando a erros internos e falsos positivos.	esforço e gerando resultados enganosos.	**escopo de automação** e um bug será reportado. O foco será em testar apenas a parte funcional do endpoint (alteração de nome).
Integridade dos Dados Iniciais (Seed): O script de seed padrão cria usuários com senhas em texto puro, em vez de hasheadas, tornando o login com esses usuário impossível.	Os dados de teste iniciais fornecidos pelo projeto não são confiáveis para testes de autenticação.	Os scripts de seed não serão utilizados para criar usuários de teste. Todos os usuários necessários (comum, admin) serão **criados dinamicamente via API** (`POST /setup/admin`) no início da suíte de testes para garantir a integridade e o estado correto dos dados. ⚠️ Parcialmente Corrigido Pós-Plano de Testes: 
Dependência de Configuração Oculta (JWT): O README do back-end omite a necessidade da variável de ambiente `JWT_SECRET` para a autenticação.	**Bloqueia totalmente** as funcionalidades de registro e login, impedindo a execução de qualquer teste que dependa de um usuário autenticado.	A variável `JWT_SECRET` foi identificada através da depuração de erros da API e adicionada manualmente ao arquivo `.env` . A documentação do back-end deve ser atualizada para incluir esta variável como um pré-requisito obrigatório. ✅ Corrigido Pós-Plano de Testes: 

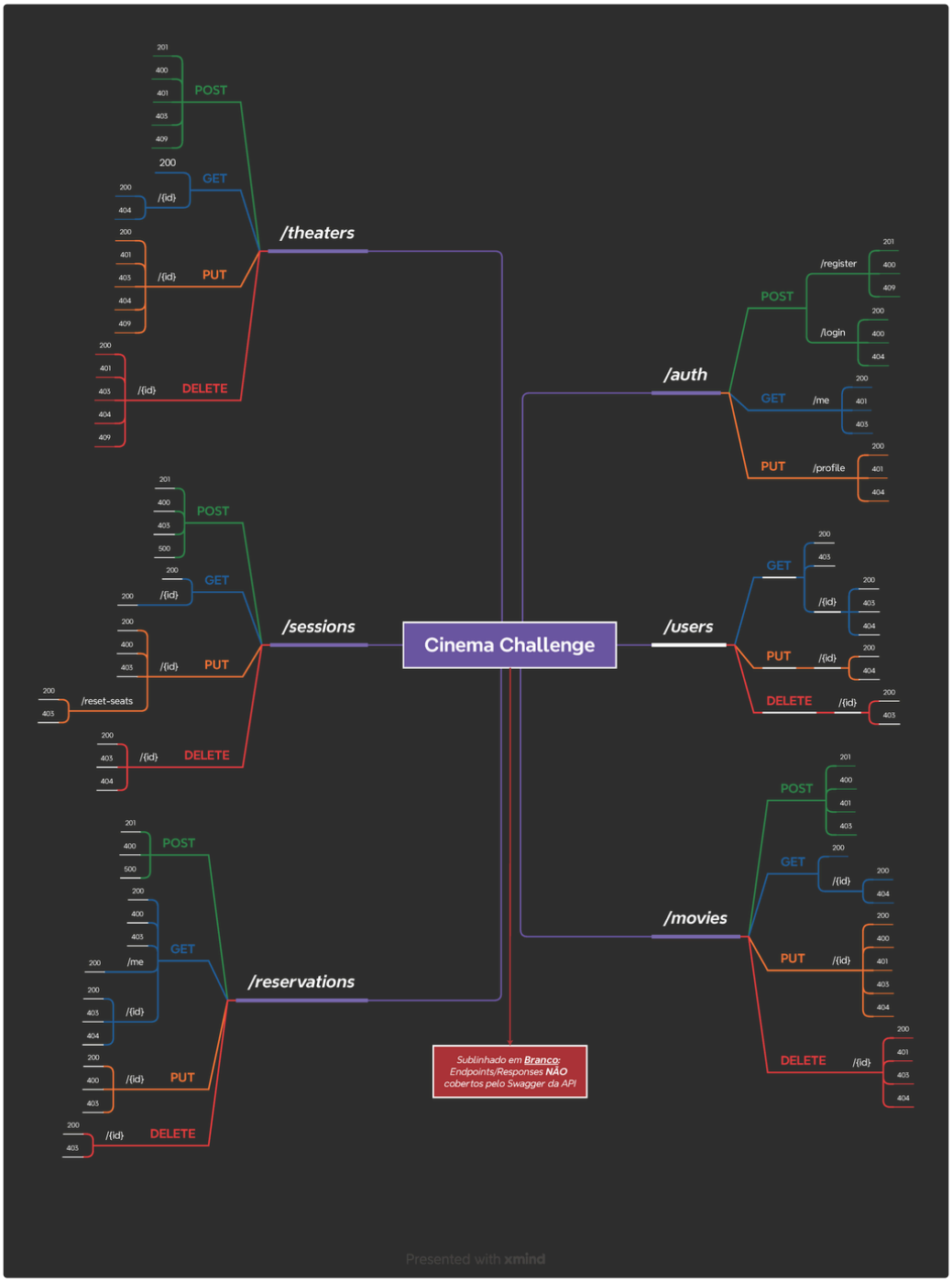
		

5.2. Análise Funcional e de Requisitos (Baseada nas User Stories)

Esta seção foca nos riscos identificados a partir da análise das próprias User Stories.

Risco Funcional / de Requisito	User Story Relacionada	Estratégia de Teste Sugerida
Ambiguidade no Fluxo Pós-Registro: O critério "redirecionado para a página de login, autenticado" é ambíguo e pode levar a múltiplas interpretações.	US-AUTH-001	Validar o comportamento real. Se o usuário precisar logar novamente, o teste de fluxo completo deve incluir esse passo. Se for autenticado automaticamente, o teste deve verificar o estado da sessão.
Responsividade do Layout: O critério de "layout responsivo" é subjetivo e pode quebrar em viewports específicos, comprometendo a usabilidade.	US-HOME-001 US-MOVIE-001 US-NAV-001	Realizar testes exploratórios manuais redimensionando a janela do navegador e usando as ferramentas de desenvolvedor para simular dispositivos móveis populares (ex: iPhone, Galaxy, iPad).

5.3. Mapa Mental da API



Sublinhado em **Branco**: Endpoints/Responses **NÃO** cobertos pelo Swagger da API

Pós Correção da Documentação:

[Doc/API] Documentação do Swagger Incompleta ou Ausente · Issue #13 · PeeeDrummm/cinema-...

📝 Resumo A documentação interativa do Swagger está severamente incompleta. Muitos endpoints (como toda a coleção /users) estão completamente ausentes, e outros...

GitHub

Drummm/cinema-challenge-tests

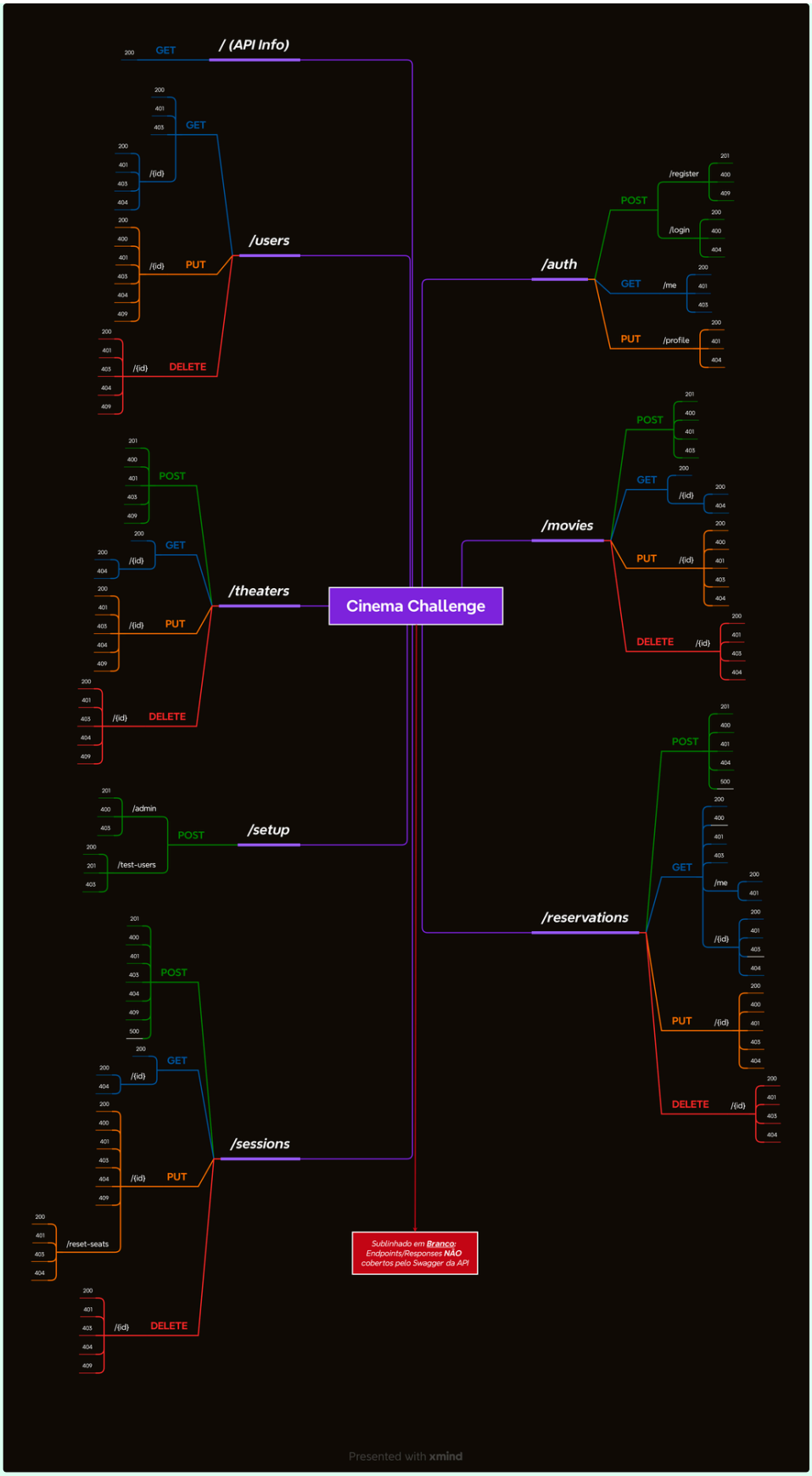
3 [Doc/API]

ocumentação do

agger Incompleta o...

omment

PeeeDrummm opened on June 25, 2025



Sublinhado em **Branco**: Endpoints/Responses **NÃO** cobertos pelo Swagger da API

6. 🛠️ Técnicas Aplicadas

Para garantir uma cobertura de testes abrangente e profunda, serão aplicadas múltiplas técnicas, combinando a eficiência da automação com a inteligência da exploração manual.

- **Automação de Testes:** Foco na criação de uma suíte de testes de regressão para validar as funcionalidades críticas da aplicação (API e UI) a cada nova alteração no código.
- **Testes Manuais Exploratórios:**
  - **Abordagem Livre:** Execução de testes não-roteirizados para descobrir bugs inesperados e entender o comportamento real do sistema.
  - **Abordagem Baseada em Heurísticas:** Aplicação de técnicas estruturadas para guiar a exploração e focar em áreas com maior probabilidade de defeitos, utilizando:
    - **Heurísticas de Nielsen:** Para identificar problemas de **usabilidade** na interface.
    - **CRUSC (Create, Read, Update, Search, Cancel/Delete):** Para validar as **operações fundamentais** do sistema para cada entidade principal (filmes, usuários, etc.).

7. 🧪 Estratégia de Teste Manual vs Automatizado

Enquanto o tópico anterior descreve as técnicas, esta seção define o **plano de aplicação** delas, seguindo o princípio da **Pirâmide de Testes** para maximizar a eficiência e a eficácia da cobertura.

- **Estratégia para o Back-end (API):**
  - **Foco:** Garantir a robustez das **regras de negócio**, a **integridade dos dados** e a **segurança**.
  - **Plano:** A maior parte dos cenários, especialmente os **negativos e de validação de dados**, será automatizada nesta camada por ser mais rápida e estável.
- **Estratégia para o Front-end (UI):**
  - **Foco:** Garantir que a **jornada do usuário** funcione de ponta a ponta e que a interface se integre corretamente com a API.
  - **Plano:** Serão automatizados apenas os **fluxos de negócio mais críticos** (caminho feliz). Testes de usabilidade e de layout serão realizados manualmente.
- **Sinergia:** O teste de UI confirma que a interface **reage corretamente** às respostas (de sucesso ou erro) que são validadas em profundidade nos testes de API.

8. ⚙️ Ferramentas e Tecnologias Utilizadas

A tabela abaixo detalha o conjunto de ferramentas e tecnologias que serão utilizadas para a execução deste plano.

Categoria	Ferramenta / Tecnologia	Finalidade
Automação de Testes	Robot Framework	Framework principal para escrever e executar os testes automatizados.
Automação de UI	Browser Library	Biblioteca do Robot para interagir com navegadores web (Front-end).

Automação de API	Requests Library	Biblioteca do Robot para fazer requisições HTTP (Back-end).
Teste Manual de API	Postman	Ferramenta para exploração, depuração e testes manuais da API.
Versionamento	Git / GitHub	Sistema para controle de versão do código de automação e para documentação.
CI/CD	GitHub Actions	Plataforma para automação da execução dos testes a cada alteração no código.
API/LLM	Google Gemini API	Validação de Responses e melhor formatação dos logs.
Editor de Código	Visual Studio Code	Ambiente de desenvolvimento para a escrita dos scripts de automação.

9. 📋 Critérios de Entrada e Saída

Esta seção define as condições que devem ser atendidas para **iniciar (Entrada)** e para considerar **concluída (Saída)** a fase de testes formais.

9.1. Critérios de Entrada (Início dos Testes)

Os testes formais e a execução dos cenários planejados só devem começar quando os seguintes critérios forem atendidos:

- **Ambiente Estável:** O back-end e o front-end da aplicação estão instalados, configurados e executando em um ambiente de teste local.
- **Acesso e Dados:** Existem **usuários de teste funcionais** (comum e admin) e há uma forma de gerar os dados necessários para os cenários (ex: via API).
- **Ferramentas Configuradas:** As ferramentas de teste (Robot Framework, Postman, etc.) estão instaladas e prontas para uso.

9.2. Critérios de Saída (Conclusão dos Testes)

A fase de testes será considerada concluída quando os seguintes critérios forem atingidos:

- **Execução dos Testes:** 100% dos casos de teste planejados (manuais e automatizados) foram **executados** pelo menos uma vez.

- **Taxa de Sucesso:** Pelo menos **95% dos casos de teste automatizados** para os fluxos críticos estão passando com sucesso.
- **Documentação de Bugs:** Todos os bugs encontrados estão **documentados e reportados**.
- **Relatório Final:** O relatório final de execução de testes (gerado pelo Robot Framework) e o resumo dos testes exploratórios estão prontos para serem apresentados.

Estes critérios de saída definem a nossa **"Definição de Concluído" (Definition of Done)** para a etapa de qualidade deste projeto.

10. 🌐 Matriz de Cenários de Teste e Priorização

A tabela a seguir consolida os **cenários de teste planejados**, sua rastreabilidade com as User Stories e a sua **priorização**, que foi definida com base no impacto para o negócio e para o usuário, e no risco técnico associado.

ID	Funcionalidade	User Story	Cenário	Prioridade	Justificativa
Autenticação					
CT001	Registro	US-AUTH-001	Criar um novo usuário com dados válidos e únicos.	**Alto**	Funcionalidade essencial para o crescimento da base de usuários.
CT002	Registro	US-AUTH-001	Tentar criar um usuário com um e-mail já existente.	**Alto**	Cenário negativo fundamental para garantir a integridade dos dados.
CT003	Registro	US-AUTH-001	Tentar criar um usuário com um formato de e-mail inválido.	**Moderado**	Validação de campo básica, importante mas geralmente tratada no front-end.
CT004	Registro	US-AUTH-001	Tentar criar um usuário com uma senha fraca.	**Moderado**	Validação de campo básica.

CT005	Login	US-AUTH-002	Realizar login com credenciais válidas.	**Muito Alto**	Cenário mais crítico da aplicação. Sem login, nenhuma funcionalidade de usuário é acessível.
CT006	Login	US-AUTH-002	Tentar realizar login com uma senha incorreta.	**Alto**	Validação de segurança básica e um dos cenários de erro mais comuns.
CT007	Login	US-AUTH-002	Tentar realizar login com um e-mail não cadastrado.	**Moderado**	Validação importante, mas de menor impacto que uma senha incorreta.
CT008	Logout	US-AUTH-003	Realizar logout e verificar se a sessão foi encerrada.	**Moderado**	Funcionalidade importante do ciclo de vida da sessão do usuário.
CT009	Logout	US-AUTH-003	Tentar acessar uma rota protegida após o logout.	**Alto**	Valida a segurança da aplicação, garantindo que a sessão foi de fato encerrada.
CT010	Perfil	US-AUTH-004	Atualizar o nome do	**Baixo**	Funcionalidade

			usuário no perfil.		secundária que não impede nenhum fluxo principal.
CT011	Perfil	US-AUTH-004	*(Baseado na Análise)* Tentar atualizar a senha.	**Baixo**	Prioridade baixa para automação, pois a funcionalidade foi identificada como inacabada/defeituosa.
Navegação e Reservas					
CT012	Home	US-HOME-001	Visualizar a página inicial, banner e filmes.	**Moderado**	Importante para a primeira impressão, mas o conteúdo é majoritariamente estático.
CT013	Lista de Filmes	US-MOVIE-001	Navegar pela lista de filmes.	**Muito Alto**	Ponto de partida para qualquer jornada de compra de ingresso.
CT014	Detalhes Filme	US-MOVIE-002	Acessar a página de detalhes de um filme.	**Muito Alto**	Passo necessário para o usuário decidir sobre a compra e ver as sessões.

CT015	Detalhes Filme	US-MOVIE-002	Tentar acessar detalhes de um filme com ID inexistente.	**Baixo**	Teste de borda, mas não é um fluxo comum.
CT016	Responsividade	US-NAV-001	*(Manual)* Verificar a responsividade do layout.	**Moderado**	Impacta a usabilidade, mas será validado manualmente.
CT017	Seleção de Assentos	US-RESERVE-001	Selecionar múltiplos assentos disponíveis em uma sessão. (Usuário logado)	**Muito Alto**	Etapa essencial e interativa do processo de reserva.
CT018	Seleção de Assentos	US-RESERVE-001	Tentar selecionar um assento que já está ocupado.	**Alto**	Regra de negócio crítica para evitar reservas duplicadas.
CT019	Seleção de Assentos	US-RESERVE-001	*(Baseado na Análise)* Validar o cálculo do subtotal.	**Alto**	Garante a precisão financeira da transação.
CT020	Checkout	US-RESERVE-002	Realizar o fluxo completo de checkout e confirmar a reserva.	**Muito Alto**	Clímax do fluxo de negócio. Representa a conversão (venda do ingresso).
CT021	Checkout	US-RESERVE-002	Tentar acessar a página de seleção de	**Alto**	Validação de segurança importante

			assentos sem estar logado.		para o fluxo principal.
CT022	Histórico	US-RESERVE-003	Acessar "Minhas Reservas" e verificar se uma nova reserva aparece.	**Alto**	Confirmação pós-venda para o usuário, completando o ciclo de confiança.
CT023	Histórico	US-RESERVE-003	Verificar se os detalhes da reserva no histórico estão corretos.	**Moderado**	Validação de integridade de dados pós-fluxo principal.
Gerenciamento (API)					
CT024	Filmes	N/A (Análise)	*(API)* Criar, listar, atualizar e deletar um filme como admin.	**Alto**	Essencial para preparação do ambiente, dado que a UI de admin não existe.
CT025	Salas	N/A (Análise)	*(API)* Criar, listar, atualizar e deletar uma sala como admin.	**Alto**	Essencial para preparação do ambiente.
CT026	Sessões	N/A (Análise)	*(API)* Criar, listar, atualizar e deletar uma sessão como admin.	**Alto**	Essencial para preparação do ambiente.
Pós-Testes Exploratórios					
CT027	Navegação (Rodapé)	US-NAV-001	Validar se o link "Filmes em Cartaz"	**Baixo**	Garante a consistência da

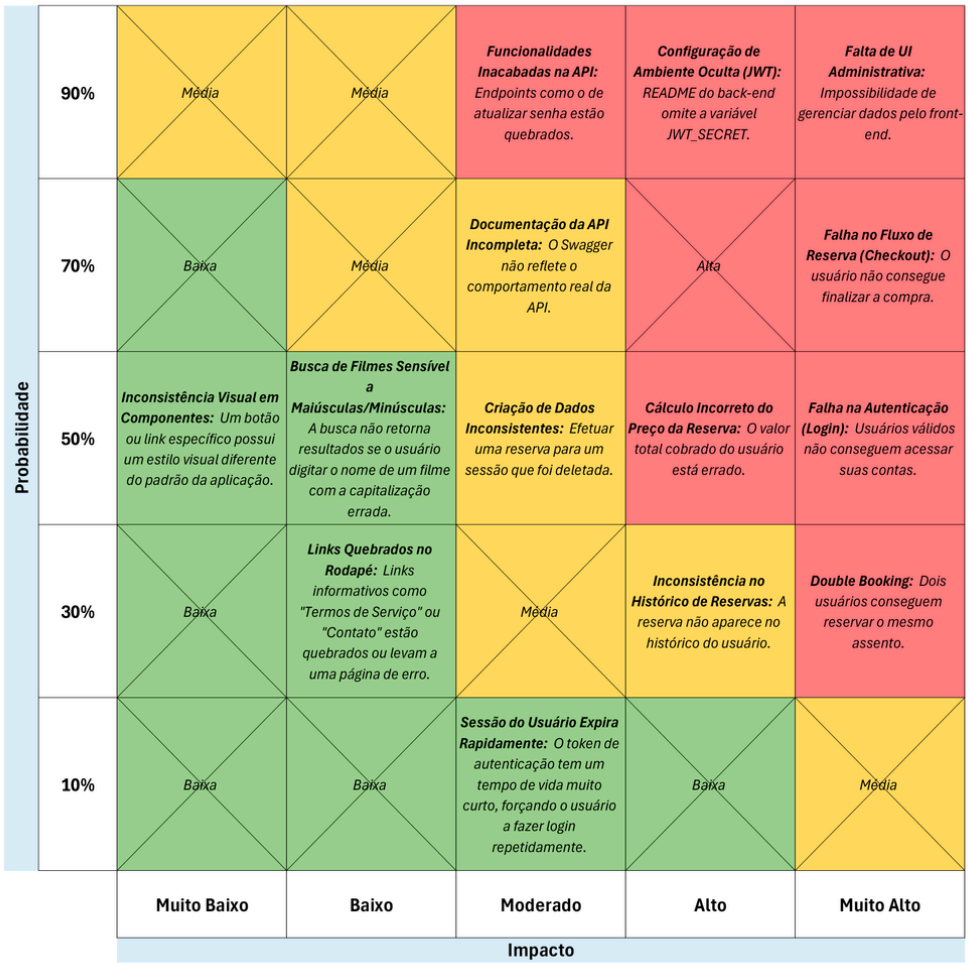
			no rodapé redireciona para a página `/movies`.		navegação e a boa experiência do usuário, evitando links quebrados.
CT028	Checkout (Não Autenticado)	US-RESERVE-002	Validar se a mensagem de erro ao tentar reservar sem login é exibida por tempo suficiente para leitura.	**Moderado**	Melhora a usabilidade, garantindo que o usuário entenda o motivo do redireciona mento para o login.
CT029	Gerenciamento (API)	N/A (Análise)	(API) Resetar assentos de uma sessão e verificar se as reservas atreladas são canceladas ou invalidadas.	**Alto**	Previne uma grave inconsistência de dados, onde um assento poderia ser vendido duas vezes.
CT030	Seleção de Assentos	US-RESERVE-001	Validar se o botão "Liberar Assentos" está visível apenas para o perfil de Administrador.	**Alto**	Valida a regra de permissão, prevenindo que usuários comuns executem ações administrativas.
CT031	Gerenciamento	N/A (Análise)	*(API)* Tentar criar uma sessão com um ID de	**Alto**	Garante a integridade referencial dos dados, prevenindo

			filme inexistente.		a criação de dados "órfãos" no banco.
CT032	Segurança	N/A (Análise)	*(API)* Tentar acessar um endpoint de admin com token de usuário comum.	**Alto**	Valida a camada de autorização (roles), um pilar da segurança do sistema.
CT033	Busca	US-MOVIE-001	Buscar um filme pelo nome e verificar se a lista é filtrada corretamente.	**Moderado**	Testa uma funcionalidade de interação chave para o usuário e garante que a busca é funcional.
CT034	Login	US-AUTH-002	Tentar logar com credenciais inválidas e validar que a mensagem de erro permanece visível.	**Alto**	Teste de regressão para um bug de UX (Bug #16) que afeta diretamente a usabilidade do login.
Pós-Reanálise das US					
CT035	Sessões	US-SESSION-001	Visualizar horários de sessão com data, hora, sala e disponibilidade.	**Muito Alto**	Fluxo inicial de reserva; usuário precisa ver opções de sessão
CT036	Navegação – Cabeçalho	US-NAV-001	Verificar presença e funcionamento do	**Alto**	Garantir acesso consistente às áreas

			cabeçalho com links em todas as páginas.		principais em todo o site
CT037	Navegação – Voltar	US-NAV-001	Navegar usando breadcrumbs ou botão “Voltar” e confirmar indicação do caminho atual.	**Moderado**	Melhora a usabilidade, dando contexto de navegação ao usuário

11. ⚠️ Matriz de Risco

Mapeamento dos riscos do projeto, combinando a probabilidade de ocorrência com o impacto no negócio, para direcionar o foco dos esforços de teste.



12. ⚙️ Testes Candidatos à Automação

Com base na análise de riscos e na priorização dos cenários, os seguintes testes foram selecionados como os principais candidatos para a automação. A seleção segue a estratégia da **Pirâmide de Testes**, focando em testes de API robustos para validar as regras de negócio e em testes de UI para garantir os fluxos críticos do usuário.

12.1. Candidatos para Automação de API (Back-end)

Estes testes formam a base da nossa suíte de regressão. São rápidos, estáveis e garantem que o "cérebro" da aplicação está funcionando corretamente.

ID do Cenário	Cenário de Teste	Justificativa para Automação
Setup & Autenticação		
CT001	Criar um novo usuário com dados válidos e únicos.	**Essencial.** Necessário para gerar dados de teste limpos e como pré-requisito para outros testes.
CT005	Realizar login com credenciais válidas.	**Crítico.** A autenticação é o portão de entrada para a maioria das funcionalidades. Garante que o acesso está funcionando e permite capturar o token para os demais testes.
CT009	Tentar acessar uma rota protegida após o logout.	Garante que a invalidação do token e a segurança das rotas estão funcionando corretamente.
Regras de Negócio e Validações		
CT002	Tentar criar um usuário com um e-mail já existente.	Valida uma regra de negócio fundamental para a **integridade dos dados**.
CT006	Tentar realizar login com uma senha incorreta.	Cenário de segurança básico e um dos fluxos de erro mais comuns.
CT018	Tentar selecionar um assento que já está ocupado.	Valida a lógica de negócio mais crítica para **evitar o double booking**.
CT029	**REMOVIDO/NAO ...** CT018 já cobre o mesmo	**Teste de regressão para um bug crítico.** Garante

	cenário. **(Bug #12)** Liberar assentos e verificar se as reservas são invalidadas.	que a falha de inconsistência de dados não retorne ao sistema.
CT031	Tentar criar uma sessão com um ID de filme inexistente.	**(Risco RSK-10)** Garante a integridade referencial da API, prevenindo a criação de dados "órfãos" no banco.
Segurança e Permissões		
CT032	Tentar acessar um endpoint de admin (ex: `POST /movies`) com token de usuário comum.	Valida a camada de **autorização (roles)**, um pilar da segurança do sistema.
Preparação do Ambiente (Admin)		
CT024 CT025 CT026	Criar Filmes, Salas e Sessões como Admin.	**Fundamental para a automação de UI.** Como a UI de admin não existe, estas keywords de API serão usadas no `Setup` dos testes de front-end para garantir que o ambiente tenha os dados necessários.

12.2. Candidatos para Automação de UI (Front-end)

Estes testes validam a jornada do usuário de ponta a ponta. São mais lentos e focam na integração e na experiência visual.

ID do Cenário	Cenário de Teste	Justificativa para Automação
Fluxos Críticos (Caminho Feliz)		
Fluxo #1	**Fluxo Completo de Reserva:** Fazer login, navegar até um filme, selecionar uma sessão, escolher um assento disponível e verificar se chegou à tela de resumo.	**O mais importante teste de UI.** Valida o fluxo de negócio que gera valor de ponta a ponta, garantindo que o usuário consegue chegar ao final do processo de compra. (Cobre os CTs `CT005`,

		`CT013`, `CT014`, `CT017`).
Fluxo #2	**Visualização do Histórico de Reservas:** Fazer login, navegar para "Minhas Reservas" e verificar se a lista de reservas é exibida.	Garante que a funcionalidade de pós-venda mais importante para o usuário está acessível e funcionando. (Cobre o CT `CT022`).
Fluxo #3	**Busca de Filmes:** Na página inicial, digitar o nome de um filme e verificar se a lista é filtrada corretamente.	**(Risco RSK-12)** Testa uma funcionalidade de interação chave e garante que a busca não é apenas visual, mas funcional.
Fluxo de Tratamento de Erro		
Fluxo #4	**Feedback de Erro no Registro:** Tentar registrar um e-mail que já existe e verificar se a mensagem de erro correspondente é exibida na tela.	Garante que a **integração entre Front-end e Back-end está funcionando** para cenários de erro e que o usuário recebe o feedback visual correto. (Cobre o CT `CT002` na camada de UI).
Fluxo #5	**Feedback de Erro para Senha Curta:** Tentar registrar um usuário com uma senha com menos de 6 caracteres e verificar se uma mensagem de erro clara é exibida na tela.	Garante que a validação de dados do front-end funciona e que o **usuário recebe feedback** claro sobre os requisitos da senha (Cobre o CT `CT004` na camada de UI).
Fluxo #6	**Tratamento de Recurso Não Encontrado (Filme):** Tentar aceder diretamente a uma URL de um filme com um ID que não existe (ex: `/movies/id_invalido`).	Valida como a UI lida com um erro `Erro ao carregar informações. Tente novamente mais tarde.` da API. A aplicação deve exibir uma página de erro amigável ou redirecionar para a home, em vez de quebrar.
Fluxo #7	**Tratamento de Recurso Não Encontrado (Sessão):** Tentar aceder diretamente	Valida como a UI lida com um erro `Erro ao`

	a uma URL de seleção de assentos para uma sessão que foi apagada.	`carregar informações. Tente novamente mais tarde.` da API. A aplicação deve exibir uma página de erro amigável ou redirecionar para a home, em vez de quebrar.

13. 🖥️ Infraestrutura de Testes

Esta seção descreve o conjunto completo de hardware, software e serviços que compõem o **ambiente onde os testes serão desenvolvidos**, **executados** e **gerenciados**.

13.1. Ambiente Local de Teste (Máquina de Execução)

- **Sistema Operacional:** Windows 10 Pro / Versão 2009 / Build 19045.5965 / 64 Bits
  - Processador: AMD Ryzen 7 1700 Eight-Core
  - Memória RAM: 16,00 GB
  - Placa de Vídeo: NVIDIA GeForce GTX 1060 6GB
- **Software de Base:**
  - Node.js (v14+): Para executar o back-end.
  - MongoDB (Atlas): Banco de dados para a aplicação.
  - Python (v3.8+): Para executar o Robot Framework.
  - Navegador Web: Microsoft EDGE (v137.0.3296.93)

13.2. Ambiente de Automação

Componentes específicos para o desenvolvimento e execução dos scripts de automação.

- **Framework de Automação:** Robot Framework.
- **Bibliotecas Principais:**
  - `BrowserLibrary`: Para automação da interface web (Front-end).
  - `RequestsLibrary`: Para automação das requisições da API (Back-end).
- **Ambiente de Desenvolvimento (IDE):** Visual Studio Code com extensões recomendadas para Robot Framework.

13.3. Ambiente de CI/CD e Versionamento

Serviços utilizados para controle de versão e automação da execução dos testes.

- **Controle de Versão:** Git, com o código do projeto de automação hospedado em um repositório no **GitHub**.
- **Render:** Utilizado para hospedar tanto o **back-end** quanto o **front-end** da aplicação, permitindo que os testes automatizados rodem contra um ambiente completo e integrado;
- **Execução Automatizada (CI/CD): GitHub Actions**, configurado para executar a suíte de testes de regressão automaticamente a cada nova alteração no código.

14. 🗓️ Cronograma de Execução e Entregáveis

Este **cronograma detalha um plano de ação diário**, garantindo que todas as atividades, desde a execução manual até a preparação da apresentação, sejam concluídas a tempo.

Data	Foco do Dia	Atividades Principais	Entregável / Meta do Dia
Semana 1: Planejamento, Exploração e Setup			
23 de jun. de 2025 24 de jun. de 2025	**Estrutura Inicial do Planejamento**	Revisar e preencher os tópicos deste documento (Plano de Testes).	**Plano de Teste v1.0.**
25 de jun. de 2025	**Teste Exploratório e Reporte de Bugs**	Executar testes exploratórios (livres e com heurísticas) no Front-end e Back-end. Documentar e criar as "Issues" no GitHub para todos os bugs e dúvidas encontrados.	**Fase de exploração concluída. Bugs e dúvidas reportados no GitHub.**
26 de jun. de 2025	**Execução dos Testes Manuais**	Seguir a "Matriz de Cenários de Teste" (tópico 10) e executar manualmente os principais cenários (positivos e negativos) no Front-end e no Back-end (via Postman).	**Validação manual dos fluxos concluída. Evidências (screenshots) coletadas para os reportes de bug.**
27 de jun. de 2025	**Documentação dos Bugs e Estrutura da Automação**	1. Criar as "Issues" no GitHub para os bugs encontrados. 2. Criar a estrutura de pastas do projeto Robot Framework e o README.md no	**Bugs mais importantes reportados no GitHub. Repositório de automação com a estrutura inicial criada.**

		repositório de testes.	
28 de jun. de 2025 29 de jun. de 2025	**Buffer / Implantação**	Colocar o back-end e o front-end na nuvem (Render ou Vercel) como preparação para o CI/CD.	**Aplicações implantadas.**
Semana 2: Automação Focada e Preparação Final			
30 de jun. de 2025	**Automação da API**	Focar em automatizar os cenários de API mais críticos: `CT001` (Registro), `CT005` (Login), `CT024` (Criar Filme), `CT032` (Acesso de Admin).	**Suíte de testes de API funcional com os principais cenários.**
1 de jul. de 2025	**Automação da UI (Fluxo Principal)**	Automatizar o "Fluxo #1" (Reserva de ponta a ponta), utilizando as keywords da API no `Setup` para criar os dados necessários (filmes, sessões).	**Principal teste de fluxo de UI (end-to-end) automatizado e funcionando.**
2 de jul. de 2025 3 de jul. de 2025	**CI/CD e Refinamento**	Configurar o workflow no GitHub Actions para rodar a suíte de testes (API e UI) automaticamente. Fazer ajustes finos nos testes existentes.	**Pipeline de CI/CD funcional. Testes estáveis.**
4 de jul. de 2025	**APRESENTAÇÃO FINAL**	Fazer uma última revisão do ambiente e dos scripts.	**Sucesso!**

15. 🔗 Integrações com CI/CD

Para garantir a qualidade contínua do projeto e a detecção rápida de regressões, a suíte de testes automatizados será integrada a uma esteira de **Integração Contínua (CI)**.

15.1. Objetivo da Integração

O objetivo principal desta integração é **automatizar a execução da suíte de testes**, fornecendo feedback rápido aos desenvolvedores sobre a qualidade das novas alterações de código.

15.2. Ferramenta Utilizada

- **GitHub Actions:** A ferramenta escolhida para a implementação da CI/CD é o GitHub Actions, por sua **integração nativa com o repositório do projeto no GitHub**, facilidade de configuração e ecossistema robusto.

15.3. Estratégia de Execução e Branching (Workflow)

A estratégia de execução seguirá um modelo de branching profissional para garantir que a branch `main` esteja sempre estável.

- **Branch `main`:** Representa a versão estável e "produção" do código. Nenhum desenvolvimento é feito diretamente aqui.
- **Branch `develop`:** É a branch principal de desenvolvimento, onde novas funcionalidades são integradas.

O workflow do GitHub Actions será acionado para proteger a branch `develop`:

- **A cada `push` para a `develop`:** A suíte de testes completa é executada para validar a integração do código.
- **A cada `Pull Request` para a `develop`:** Antes que uma nova feature (de uma `feature-branch`) possa ser mesclada na `develop`, os testes são executados como uma verificação obrigatória. O merge só é permitido se os testes passarem.

15.4. Processo do Workflow Automatizado

O workflow executará os seguintes passos em um ambiente limpo e controlado fornecido pelo GitHub Actions:

1. **Checkout:** O código do projeto de automação é baixado para o ambiente de execução.
2. **Setup do Ambiente:** O ambiente é preparado com a versão correta do Python.
3. **Instalação de Dependências:** As bibliotecas necessárias para a automação (Robot Framework, BrowserLibrary, RequestsLibrary, etc.) são instaladas a partir do arquivo `requirements.txt`.
4. **Execução dos Testes:** O comando `robot` é executado para rodar a suíte de testes completa (API e UI).
5. **Relatório:** Se um teste falhar, o workflow é interrompido e marcado como "falho". Se todos passarem, o workflow é marcado como "sucesso".

16. 📊 Cobertura de Testes

A cobertura de testes apresentada nesta seção foi calculada com base nos critérios definidos no artigo ***Test Coverage Criteriafor RESTful Web APIs***. Ela representa os itens técnicos testados ao menos uma vez durante a elaboração dos cenários execução parcial da suíte.

A tabela abaixo consolida a cobertura funcional e técnica obtida até o momento:

Critério	Total documentado	Coberto (manual + autom.)	% Cobertura	Observações

Path Coverage (input)	**22 paths** – 7 módulos (`/auth` , `/users` , `/movies` , `/theaters` , `/sessions` , `/reservations` , `/setup`)	17	77%	Faltam cenários GET. para `/sessions` com filtros, POST `/setup/*` em produção.
Operator Coverage (input)*(métodos HTTP)*	32 operações (GET/POST/PUT/DELETE/PUT-custom) mapeadas nos 7 módulos acima	23	72%	Endpoints de build-only (`/setup`) e alguns DELETE admin ainda sem teste negativo.
Parameter Coverage (input)	49 parâmetros de body/query/path identificados nos contratos Swagger	25	51%	Falta cobrir query-params avançados (`sort` , `page` , `limit`) e path `id` nos fluxos DELETE/PUT.
Parameter-Value Coverage (input)	88 valores esperados (válidos + inválidos)	38	43%	∅ casos de **boundary** (id mal-formado, limite = 0, data no passado).
Content-Type Coverage	2 (JSON + multipart)	2	100 %	JSON default e `multipart/form-data` (upload de pôster).
Operation-Flow Coverage	10 fluxos de negócio identificados	7	70%	Fluxos pendentes: redefinição de assentos (/reset-seats) e cancelamento de reserva pago.
Response-Body Properties	14 chaves críticas (token, pagination, seats, etc.)	7	50%	Cobrir campos aninhados em `data.*` e erros padronizados.
Status-Code Coverage	9 códigos documentados (200, 201, 204, 400, 401, 403, 404, 409, 422)	7	78%	Falta validar 204 (delete vazio) e 422 (validação de schema).

16.1 Distribuição dos 28 Bugs Abertos

Severidade / Prioridade	Quantidade	%	Camada mais afetada	Observação
Crítica	4	14%	Back-end/API	Falhas de segurança e lógica de reserva.
Alta	6	21%	Front-end & Dados	Layout crítico, permissões, estado de auth.
Moderada	12	43%	Front-end (UX/UI)	Erros de mensagem, validação de campos, responsividade.
Baixa	2	7%	Navegação/UI	Links quebrados, ícones sobrepostos.
Pergunta / Feature	4	14%	Requisitos	Gaps de requisito que ainda viram teste quando definidos

17. 📦 Artefatos Gerados (Entregáveis do Projeto)

Os seguintes artefatos serão produzidos como resultado do processo de teste e representam os entregáveis deste desafio.

17.1. Documentação de Planejamento e Análise

1. **Plano de Testes:**
   - **Descrição:** Este próprio documento, detalhando a estratégia, escopo, riscos e planejamento completo do esforço de teste.
2. **Collection do Postman:**
   - **Descrição:** Um conjunto de requisições organizadas no Postman, utilizado para a exploração manual da API. Serve como uma documentação viva e executável do comportamento do back-end.

17.2. Código-Fonte e Automação

3. **Repositório do Projeto de Automação no GitHub:**
   - **Descrição:** Contém todo o código-fonte da automação de testes, incluindo: GitHub - PeeeDrummm/cinema-challenge-tests: Repositório do código de testes e report de Bugs/Melhorias
     - A estrutura de pastas organizada.
     - Os scripts de teste em Robot Framework para a API e para a UI.
     - O arquivo `README.md` com as instruções de instalação e execução.
4. **Workflow de CI/CD (GitHub Actions):**

- **Descrição:** O arquivo de configuração (`.yml`) que define a esteira de integração contínua, permitindo a execução automática dos testes.

17.3. Resultados e Evidências de Qualidade

5. **Relatórios de Execução Manual:**
   - **Descrição:** A planilha `Planilha de Execução de Testes.xlsx` preenchida após os testes manuais ROTERIZADOS. cinema-challenge-tests/Planilha de Execução de Testes.xlsx at main · PeeeDrummm/cinema-challenge-tests
6. **Sessão de Teste Exploratório**:
   - **Descrição:** O documento `Sessão de Teste Exploratório.docx` preenchido após os testes manuais exploratórios livres e com heurística. cinema-challenge-tests/Sessão de Teste Exploratório.docx at main · PeeeDrummm/cinema-challenge-tests
7. **Relatórios de Execução Automatizada:**
   - **Descrição:** Os arquivos `report.html` e `log.html` gerados pelo Robot Framework a cada execução. https://peeedrummm.github.io/cinema-challenge-tests/report.html
8. **Reportes de Bug:**
   - **Descrição:** Documentação formal de cada falha ou inconsistência encontrada (ex: como "Issues" no GitHub). Cada reporte contém título, passos para reproduzir, resultado esperado vs. atual, e evidências (screenshots). https://github.com/PeeeDrummm/cinema-challenge-tests/issues Conectar a conta do Github

18. 📄 Interpretação dos Resultados

Durante a execução dos testes planejados, **a análise foi além do resultado sucesso ou falha**. O objetivo central foi **interpretar profundamente o que esses resultados significam para a qualidade geral da aplicação**, a experiência final do usuário e os riscos potenciais para o negócio.

18.1. Resumo dos Cenários Executados

- **24 testes passaram com sucesso**, abrangendo fluxos críticos como autenticação válida, reserva completa, histórico de reservas, CRUDs administrativos e validações básicas.
- **14 testes apresentaram falhas**, destacando principalmente problemas relacionados à segurança (login incorreto), usabilidade (feedbacks rápidos e layout instável), inconsistências em mensagens de erro e problemas de configuração inicial.

18.2. Impacto para a Aplicação

- A aplicação encontra-se em um estado **parcialmente estável nos principais fluxos de negócio**, especialmente nas **rotas felizes de reserva e autenticação válida**. Contudo, as falhas detectadas apontam fragilidades significativas em áreas sensíveis como segurança, clareza nas mensagens de erro, integridade de dados e configuração inicial. Esses pontos afetam diretamente a confiança do usuário, podendo comprometer seriamente a experiência e gerar situações de risco como reservas duplicadas e abandono por dificuldades de uso.

18.3. Áreas de Risco Identificadas

- **Segurança e autenticação:** Falhas relacionadas à validação de login inválido e gerenciamento inadequado de tokens JWT aumentam significativamente a vulnerabilidade do sistema (CT006, CT007, CT034).

- **Permissões administrativas e integridade dos dados:** Problemas como exposição de funcionalidades restritas e double booking de assentos podem causar impactos operacionais graves (CT029, CT030).
- **Experiência do usuário (UX/UI):** Feedback visual inadequado, mensagens de erro que desaparecem rapidamente, layout quebrado e responsividade insuficiente afetam negativamente a experiência (CT004, CT011, CT012, CT014, CT016, CT021, CT027, CT028).
- **Configuração e integridade inicial dos dados:** A falta de arquivos essenciais (imagens) no projeto compromete a qualidade da apresentação inicial (CT012).

18.4. Recomendações

- **Priorizar correção dos defeitos críticos** relacionados à segurança, permissões e integridade dos dados, especialmente os identificados pelos CT006, CT007, CT029 e CT030.
- **Revisar detalhadamente as mensagens de erro e o tempo de exibição** (CT028, CT034) para garantir uma comunicação eficaz com o usuário final.
- Melhorar urgentemente os **layouts e a responsividade das páginas críticas** (CT012, CT014, CT016).
- Resolver problemas de configuração inicial, especialmente relacionados aos dados e imagens ausentes no **ambiente de seed** (CT012).
- **Executar testes adicionais focados em cenários negativos** e de limites para robustecer o sistema diante de dados inválidos ou incompletos.

18.5. Conclusão

- A rodada de testes demonstrou que **o sistema atende parcialmente aos critérios essenciais de qualidade**, apresentando **risco moderado a alto** em áreas específicas. As correções propostas são urgentes e cruciais, devendo ser implementadas rapidamente para garantir a integridade operacional e melhorar significativamente a confiança dos usuários e stakeholders. A automação já demonstrou grande valor na identificação precoce dos principais defeitos, indicando que a expansão e aprimoramento contínuo da cobertura automatizada trarão ganhos substanciais para o projeto.