Plano de Teste

Projeto: New Cinema App 1.0.0 (API)

1. Objetivo

Garantir a qualidade da aplicação, validando as regras de negócio e o funcionamento das APIs REST, de modo que atendam aos requisitos especificados e mantenham a integridade.

2. Escopo

Os testes contemplarão as funcionalidades essenciais da aplicação expostas pelas APIs REST, com foco nos fluxos críticos para o negócio.

Recursos a serem Testados:

- **/auth**
  - POST
    - Fazer Login
    - Fazer Cadastro
- **/movies**
  - POST
    - Cadastrar um novo filme
  - PUT
    - Editar um filme
  - DELETE
    - Deletar um filme
- **/reservations**
  - POST
    - Criar uma nova reserva
  - PUT
    - Editar uma reserva
  - DELETE
    - Deletar uma reserva
- **/sessions**
  - POST
    - Criar uma nova sessão
  - PUT
    - Editar uma sessão
  - DELETE
    - Deletar uma sessão
- **/theaters**
  - POST
    - Criar uma sala de cinema
  - PUT
    - Editar uma sala de cinema
  - Delete
    - Deletar uma sala de cinema

3. Análise

Visão Geral do Produto

O Cinema APP é uma API REST voltada para o gerenciamento de cinemas e reservas. A aplicação oferece funcionalidades para dois tipos de perfis:

- **Visitante:** Permite visualizar filmes e buscar sessões
- **Clientes**: cadastro, autenticação, busca de sessões e realização de reservas.
- **Administrador:** gerenciar cinemas, filmes, sessões e usuários.

Foco dos Testes:

Os testes terão com objetivo principal:

Garantir que as **regras de negócios** sejam respeitadas, como:

- Validação de login e autenticação.
- Controle de assentos disponíveis por sessão.
- Consistência em criar, editar e remover filmes, sessões e cinemas.

4. Metodologias:

- Mapa Mental

Estratégia de Teste:

- Teste de validação de regras de negócio
- Heurística CRUD

Recursos:

- **Equipe:** 1 Bolsista de Quality Engineer
- **Ferramentas:** Postman, RobotFrameWork

5. Cenários de testes planejados

Esta seção descreve as funcionalidades que serão validadas durante as atividades de teste, com base nas histórias de usuário fornecidas pelo repositório. Os casos de teste foram elaborados a partir dessas histórias, cobrindo tanto cenários positivos quanto negativos, sempre que aplicável.

ID	História de Usuário	Funcionalidade Testada	Critérios de Aceitação
US-AUTH-001	Registro de Usuário	Cadastro de novos usuários	Validação de e-mail e senha, prevenção de duplicidade, redirecionamento após sucesso
US-AUTH-002	Login de Usuário	Acesso com credenciais válidas	Autenticação JWT, redirecionamento para home
US-AUTH-003	Logout de Usuário	Encerramento de sessão	Remoção de token e bloqueio de rotas protegidas
US-AUTH-004	Gerenciar Perfil	Edição de dados do perfil	Atualização de nome e confirmação visual de sucesso

ID	História de Usuário	Funcionalidade Testada	Critérios de Aceitação
US-RESERVE-001	Selecionar Assentos	Seleção de assentos no mapa	Controle de disponibilidade e subtotal dinâmico
US-RESERVE-002	Processo de Checkout	Finalização da compra	Seleção de método de pagamento e confirmação visual
US-RESERVE-003	Visualizar Minhas Reservas	Histórico de reservas do usuário	Exibição de cards com detalhes e status da reserva

6. Priorização

Do ponto de vista de negócios, falhas nestes fluxos podem impactar diretamente a receita, a experiência do usuário e a credibilidade do sistema. Dado essa análise os testes foram priorizados da seguinte forma.

Alta Prioridade:

- **Clientes/Visitantes**
  - Cadastro de Clientes
  - Login de Clientes
- **Filmes**
  - Criar, Editar, Deletar Filme
  - Buscar filme por ID
- **Reservas**
  - Criar, Editar, Deletar Reserva
  - Listar Reserva do usuário logado
  - Buscar reserva por ID
- **Sessões**
  - Criar, Editar, Deletar Sessão
  - Buscar Sessão por ID
- **Cinemas**
  - Criar, Editar, Deletar Cinema

Média Prioridade:

- **Clientes/Usuários**
  - Buscar usuário por ID
- **Sessões**
  - Limpar assentos de uma sessão

Baixa Prioridade:

- **Usuários**
  - Listar todos usuários cadastrados
- **Filmes**
  - Listar todos filmes cadastrados
- **Reservas**
  - Listar todas reservas cadastradas
- **Sessões**
  - Listar todas sessões cadastradas
- **Cinema**
  - Listar todos cinemas cadastrados

7. Matriz de risco

RISCO	Probabilidade	Impacto	Fator de Risco
API permite criação de sessões duplicadas sem validação de conflito	5	5	25
Campo paymentStatus não é atualizado ao editar reserva	4	5	20
Cadastro aceita nome de sala de cinema duplicado	4	4	16
Edição de sala de cinema permite nome duplicado	4	5	20
Falha de autorização não tratada corretamente (cliente acessa recursos de admin)	3	5	15
Falha ao validar campos obrigatórios em criação de filme	3	4	12
API não valida token de autenticação em algumas rotas	3	5	15

8. Cobertura de testes

Operator Coverage (input)

Confere a cobertura de testes de todos os métodos existentes na API REST (GET, POST, PUT, DELETE…).

Endpoints automatizados:

***/auth/register* - POST**

CT001.001 - Cadastrar com um e-mail e senha válidos
CT001.002 - Cadastrar com um e-mail inválido
CT001.003 - Cadastrar com campo em branco
CT001.004 - Cadastrar com email duplicado
CT001.005 - Cadastrar um cliente com e-mail de administrador

***/auth/login* - POST**

CT002.001 - Fazer Login com dados válidos
CT002.002 - Fazer Login com dados não cadastrados
CT002.003 - Fazer login com e-mail inválido

/users/:id - **DELETE**

***/movies* - POST**

CT003.001 - Cadastrar um filme válido
CT003.002 - Cadastrar um filme com campos em branco
CT003.003 - Cadastar um filme sem token de autenticação
CT003.004 - Cadastrar um filme com token de cliente

***/movies/:id* - PUT**

CT004.001 - Editar um filme cadastrado
CT004.002 - Editar um filme utilizando token de cliente
CT004.003 - Editar um filme sem token de autorização

***/movies/:id* - DELETE**

CT005.001 - Deletar um filme cadastrado
CT005.002 - Deletar um filme sem token de autorização
CT005.003 - Deletar um filme com token de cliente

***/theaters* - POST**

CT006.001 - Criar uma Sala de cinema válida
CT006.002 - Criar uma sala de cinema com nome duplicado
CT006.003 - Criar uma sala de cinema sem token de autorização
CT006.004 - Criar sala de cinema com token de cliente

/theaters/:id - PUT

CT007.001 - Editar uma Sala de Cinema com dados válidos
CT007.002 - Editar uma sala de cinema sem token de autorização
CT007.003 - Editar uma sala de cinema com token de cliente
CT007.004 - Editar uma sala e preencher com um nome já existente

/theaters/:id - DELETE

CT008.001 - Deletar uma Sala de Cinema válida
CT008.002 - Deletar uma sala de cinema sem token de autorização
CT008.003 - Deletar uma sala de cinema com token de cliente

***/sessions* - POST**

CT009.001 - Criar uma nova sessão
CT009.002 - Criar uma sessão sem token de autorização
CT009.003 - Criar sessão com token de cliente
CT009.004 - Criar sessão duplicada

***/sessions/:id* - PUT**

CT010.001 - Editar uma sessão cadastrada
CT010.002 - Editar uma sessão sem token de autorização
CT010.003 - Editar uma sessão com token de cliente

***/sessions/:id* - DELETE**

CT011.001 - Deletar uma sessão válida
CT011.002 - Deletar uma sessão sem token de autorização
CT011.003 - Deletar uma sessão com token de cliente

***/reservations* - POST**

CT012.001 - Criar uma reserva válida
CT012.002 - Criar uma reserva sem token de autorização
CT012.004 - Criar uma reserva duplicada

***/reservations/:id* - PUT**

CT013.001 - Editar status de uma reserva válida
CT013.002 - Editar status de uma reserva sem token de autorização
CT013.003 - Editar status de uma reserva com token de cliente

***/reservations/:id* - DELETE**

CT014.001 - Deletar uma reserva válida
CT014.002 - Deletar uma reserva sem token de autorização
CT014.003 - Deletar uma resrva com token de cliene

Métrica	**Valor**
Total de Endpoints	28
Endpoints Automatizados	15
Cobertura de Operadores (Operator Coverage)	**≈ 53,6%**

9. Testes candidatos à automação

Os seguintes testes ainda não foram automatizados, mas são considerados bons candidatos à automação futura, pois envolvem fluxos críticos da aplicação, verificações frequentes de segurança/autorização e operações de consulta amplamente utilizadas pelos usuários.

/auth

GET /auth/me – Retorna o perfil do usuário autenticado.

- importante para validar o fluxo de autenticação e controle de sessão.

/reservations

GET /reservations/me – Retorna reservas do usuário autenticado
/reservations/id – Retorna detalhes de uma reserva.

- garantem a consistência de dados e o controle de acesso entre clientes e administradores

/sessions

GET /sessions/:id – Retorna detalhes de uma sessão
/sessions – Retorna todas as sessões.

- garantem a consistência de dados e o controle de acesso entre clientes e administradores

10. Mapeamento de Issues

Durante a execução dos **testes automatizados**, foram identificados alguns comportamentos inconsistentes em relação às regras de negócio e respostas esperadas da API. As issues abaixo foram registradas para acompanhamento, correção e revalidação.

- **ID da Issue:** 001
- **Título/Resumo:** Login com credenciais inválidas retorna status code 401 incorreto
- **Autor/Relator:** Rafael Soares
- **Data de Registro:** 03/10/25
- **Status:** Aberto
- **Versão do Software:** 1.0.0
- **Ambiente de Teste:** Desenvolvimento

2. Descrição Detalhada do Defeito

- **Descrição:**- Ao realizar uma requisição de login via API (POST /auth/login) com credenciais inválidas, o sistema retorna o status code 401 (Unauthorized) quando o esperado seria 400 (Bad Request).

- **Resultado Esperado:** A API deve retornar a mensagem de credenciais inválidas com o **Status Code 400**.

```
{
"success": false,
"message": "Invalid credentials"
 }
```

- **Resultado Real:** A API retornou a mensagem de falha, porém com o Status Code 401 de não autorizado o que não é correto

```
{
"success": false,
"message": "Invalid credentials"
}
```

3. Passos para Reprodução

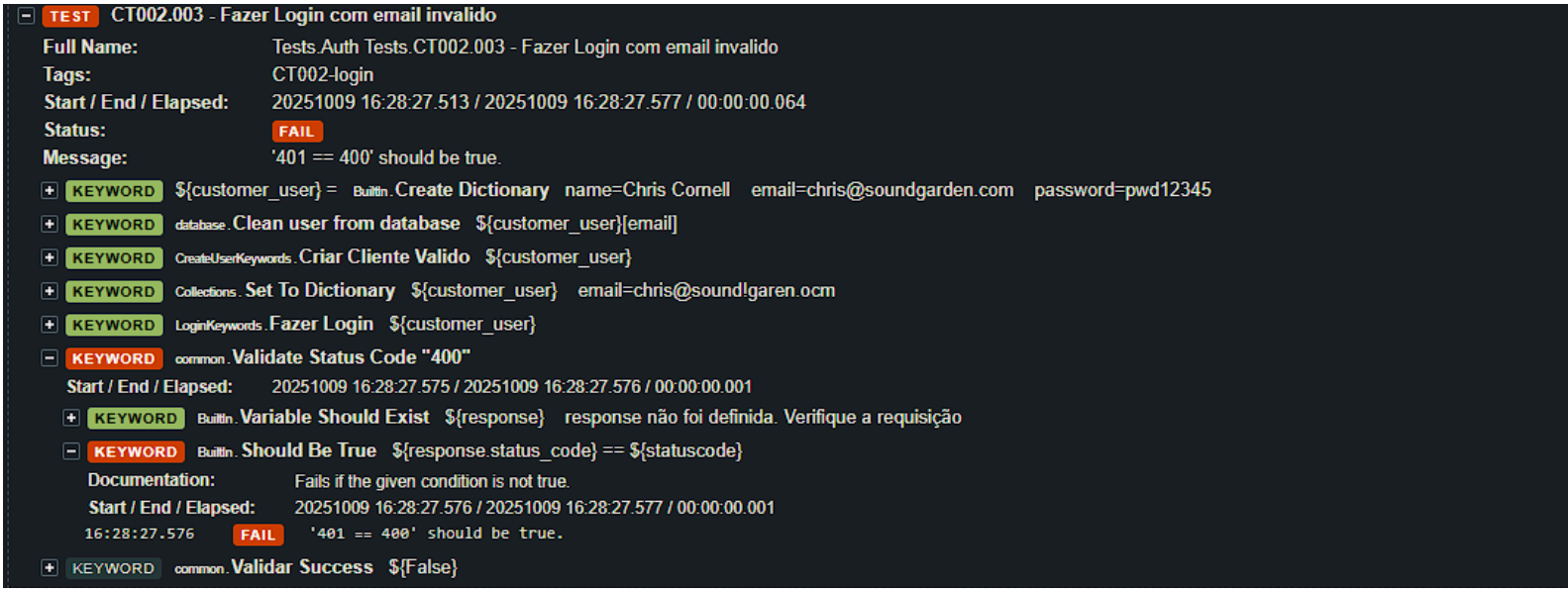
Liste os passos exatos para reproduzir o defeito, incluindo:

[Passo 1] Enviar requisição POST para `/auth/login` com os seguintes campos:

```
{  "email": "exampleInvalid@E!#mail.cmo",
           "password": "senhna123"}
```

[Passo 2] Observar o Status Code retornado

4. Evidências



5. Classificação da Issue

- **Prioridade:** Baixa
- **Gravidade/Impacto:** Baixa
- **Frequência:** Ocorre consistentemente
- **Tipo de Defeito:** Erro técnico / implementação

- **ID da Issue:** 002
- **Título/Resumo:** Requisição PUT não está editando todos campos de uma reserva.
- **Autor/Relator:** Rafael Soares
- **Data de Registro:** 03/10/25
- **Status:** Aberto
- **Versão do Software:** 1.0.0
- **Ambiente de Teste:** Desenvolvimento

2. Descrição Detalhada do Defeito

- **Descrição:**

  Ao executar uma requisição PUT para editar uma reserva via API (PUT /reservations/{id}), o endpoint não atualiza corretamente todos os campos enviados no payload. Especificamente, o campo paymentStatus não é atualizado no banco de dados, enquanto o campo status é processado corretamente.

- **Resultado Esperado:** A API deve atualizar os campos status e paymentStatus no banco de dados corretamente e retornar status Code 200.

REQUEST


```
{
 "status": "confirmed",
 "paymentStatus": "completed"
}
```


RESPONSE


```
{
  "success": true,
  "data": {
     "_id": "60d0fe4f5311236168a109ce",
     "user": "60d0fe4f5311236168a109ca",
      "session": "60d0fe4f5311236168a109cd",
     "seats": [
      {
        "row": "C",
       "number": 5,
       "type": "full"
          },
          {
       "row": "C",
       "number": 6,
       "type": "half"

       }
     ],
```

```
"totalPrice": 30,
"status": "confirmed",
"paymentStatus": "completed",
"paymentMethod": "credit_card",
"paymentDate": "2021-06-21T12:30:00.000Z",
"createdAt": "2021-06-21T12:00:00.000Z" }
```


- **Resultado Real:** A API atualizou apenas o campo status e não fez alterações no campo paymentStatus no banco de dados e retornou status code 200

REQUEST


```
{
 "status": "confirmed",
 "paymentStatus": "completed"
}
```


RESPONSE


```
{
  "success": true,
  "data": {
     "_id": "60d0fe4f5311236168a109ce",
     "user": "60d0fe4f5311236168a109ca",
      "session": "60d0fe4f5311236168a109cd",
     "seats": [
      {
        "row": "C",
       "number": 5,
       "type": "full"
         },
         {
       "row": "C",
       "number": 6,
       "type": "half"

       }
     ],
         "totalPrice": 30,
         "status": "confirmed",
         "paymentStatus": "pending",
         "paymentMethod": "credit_card",
         "paymentDate": "2021-06-21T12:30:00.000Z",
         "createdAt": "2021-06-21T12:00:00.000Z" } }
```

3. Passos para Reprodução

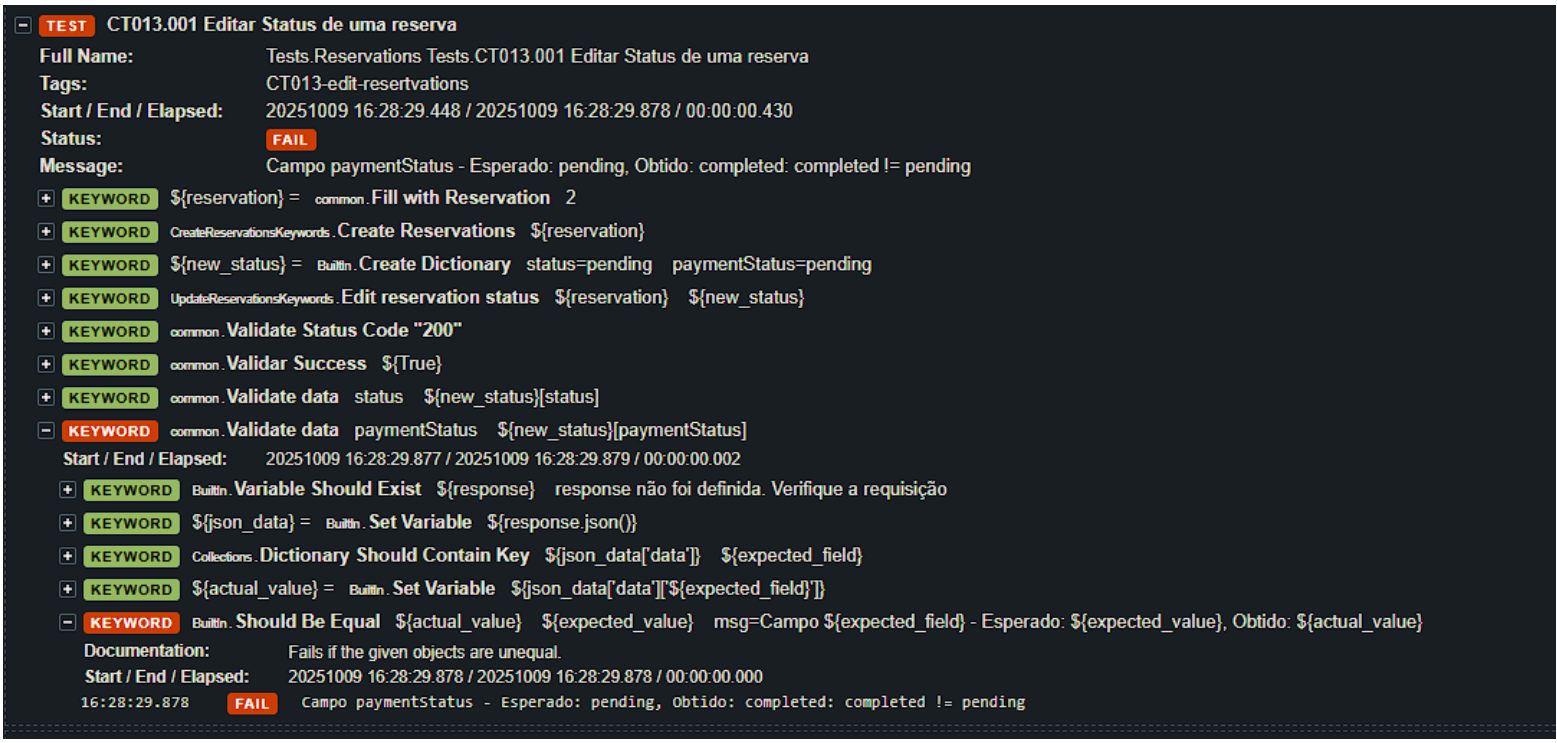
Liste os passos exatos para reproduzir o defeito, incluindo:

[Passo 1] Enviar requisição PUT para `/reservations/{id}` com os seguintes campos:

```
{
 "status": "confirmed",
 "paymentStatus": "completed"
}
```

[Passo 2] Observar o Status Code retornado

4. Evidências



5. Classificação da Issue

- **Prioridade:** Alta
- **Gravidade/Impacto:** Alta
- **Frequência:** Ocorre consistentemente
- **Tipo de Defeito:** Lógica de Negócio

- **ID da Issue:** 003
- **Título/Resumo:** API permite criação de sessões duplicadas sem validação de conflito
- **Autor/Relator:** Rafael Soares
- **Data de Registro:** 03/10/25
- **Status:** Aberto
- **Versão do Software:** 1.0.0
- **Ambiente de Teste:** Desenvolvimento

2. Descrição Detalhada do Defeito

- **Descrição:** Ao criar uma nova sessão via API (POST /sessions), o sistema não realiza a validação de conflito de horários e permite a criação de sessões duplicadas para o mesmo cinema no mesmo horário, violando a regra de negócio.

- **Resultado Esperado:** A API deve retornar a mensagem de conflito com o **Status Code 409**.

```
{
"success": false,
"message": "Session conflict (time overlap)"
 }
```

- **Resultado Real:** A API retorna a mensagem de sucesso criando a uma sessão duplicada


```
{"success": true,
"data": {
"_id": "60d0fe4f5311236168a109cd",
"movie": "60d0fe4f5311236168a109cb",
"theater": "60d0fe4f5311236168a109cc",
"datetime": "2025-06-20T18:30:00.000Z",
"fullPrice": 20,
 "halfPrice": 10,
 "seats": [ {
   "row": "A",
   "number": 1,
    "status": "available"
},
{
"row": "A",
```

```
      "number": 2,
           "status": "reserved"}],
"createdAt":"2021-06-21T12:00:00.000Z"} }
```

3. Passos para Reprodução

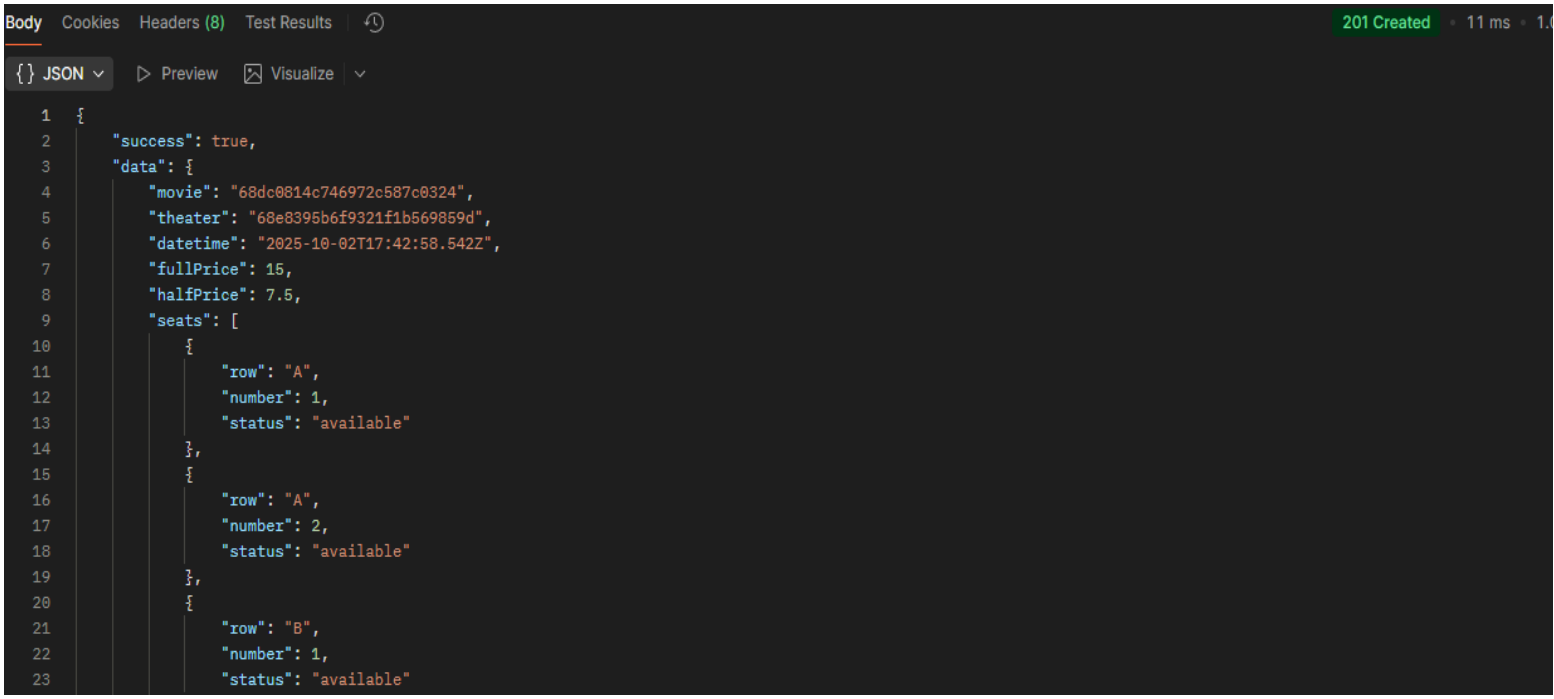
Liste os passos exatos para reproduzir o defeito, incluindo:

[Passo 1] Enviar requisição POST para `/sessions` com os seguintes campos:

```
{     "movie": "{movie_id}",
      "theater": "{theater_id}",
      "datetime": "2025-10-09T22:46:25.883Z",
      "fullPrice": 0,
      "halfPrice": 0 }
```

[Passo 2] Observar a resposta da requisição

4. Evidências



5. Classificação da Issue

- **Prioridade:** Alta
- **Gravidade/Impacto:** Alta
- **Frequência:** Ocorre consistentemente
- **Tipo de Defeito:** Lógica de Negócio

- **ID da Issue:** 004
- **Título/Resumo:** Criar uma sala de cinema duplicada retorna o status code incorreto
- **Autor/Relator:** Rafael Soares
- **Data de Registro:** 03/10/25
- **Status:** Aberto
- **Versão do Software:** 1.0.0
- **Ambiente de Teste:** Desenvolvimento

2. Descrição Detalhada do Defeito

- **Descrição:**

  Ao tentar criar uma sala de cinema com nome duplicado através da API (POST /theaters), o sistema retorna status code 400 (Bad Request) quando o comportamento esperado conforme documentação seria status code 409 (Conflict).

- **Resultado Esperado:** A API deve retornar a mensagem indicando que já existe uma sala de cinema com o nome preenchido com o **Status Code 409**.

```
{
"success": false,
"message": "Theater with that name already exists"
 }
```

- **Resultado Real:** A API retornou a mensagem de falha, porém com o **Status Code 400** o que não condiz com a documentação.

```
{"success":false,"message":"Duplicate field value entered","field":{"name":"Sala - 90"}}
```

3. Passos para Reprodução

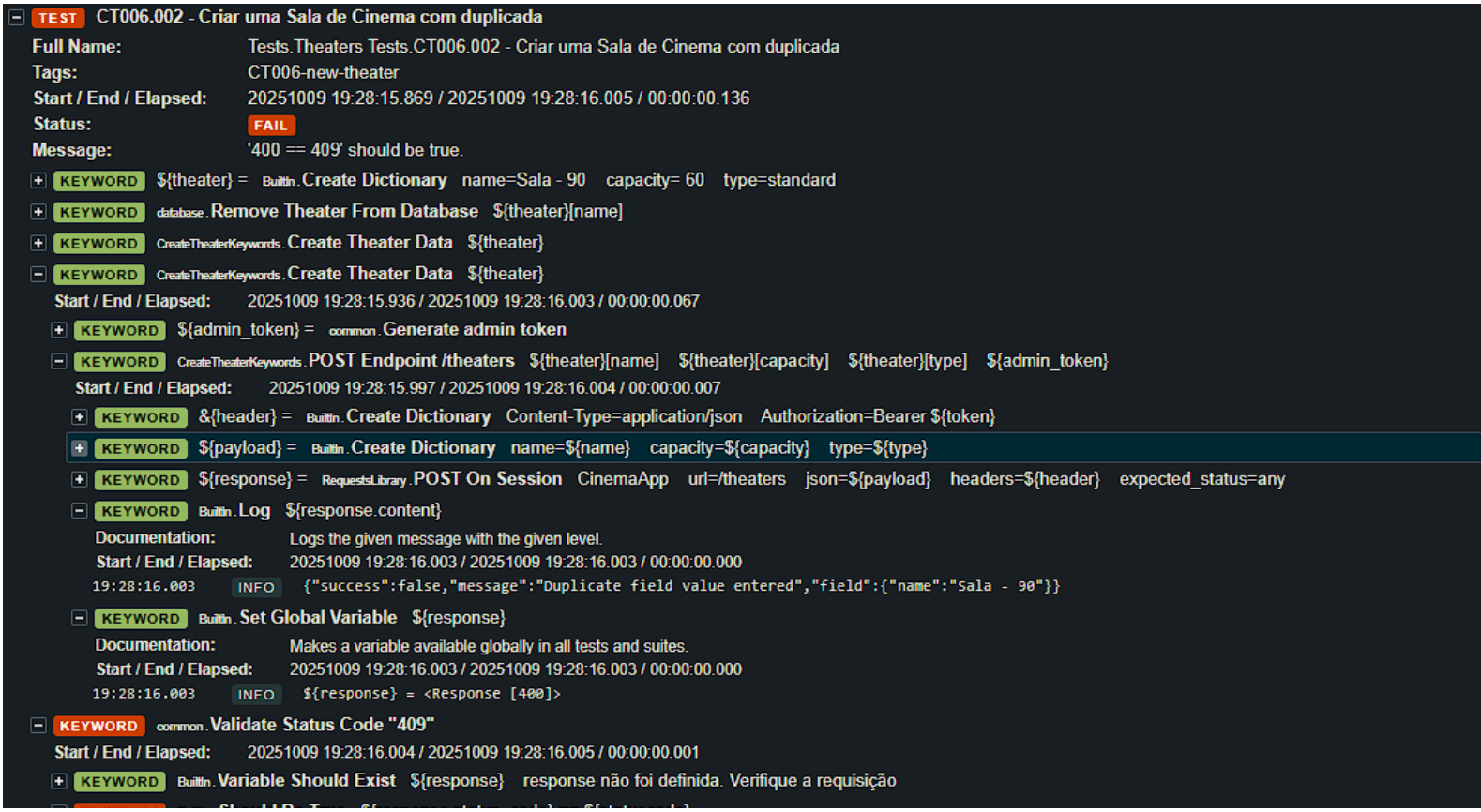
Liste os passos exatos para reproduzir o defeito, incluindo:

[Passo 1] Enviar requisição POST para `/theaters` com os seguintes campos:

```
"name": "string",
"capacity": 1,
"type": "standard"}
```

[Passo 2] Observar o Status Code retornado

4. Evidências



5. Classificação da Issue

- **Prioridade:** Baixa
- **Gravidade/Impacto:** Baixa
- **Frequência:** Ocorre consistentemente
- **Tipo de Defeito:** Erro técnico / implementação

- **ID da Issue:** 005
- **Título/Resumo:** Edição de sala de cinema com nome duplicado retorna status code 400 incorretamente
- **Autor/Relator:** Rafael Soares
- **Data de Registro:** 03/10/25
- **Status:** Aberto
- **Versão do Software:** 1.0.0
- **Ambiente de Teste:** Desenvolvimento

2. Descrição Detalhada do Defeito

- **Descrição:**
  Ao tentar editar uma sala de cinema atribuindo um nome que já existe no sistema através da API (PUT /theaters/{id}), o endpoint retorna status code 400 (Bad Request) quando o comportamento esperado conforme documentação seria status code 409 (Conflict).

- **Resultado Esperado:** A API deve retornar a mensagem indicando que já existe uma sala de cinema com o nome preenchido com o **Status Code 409**.

```
{
"success": false,
"message": "Theater name already in use"
 }
```

- **Resultado Real:** A API retornou a mensagem de falha, porém com o **Status Code 400** o que não condiz com a documentação.

```
{"success":false,"message":"Duplicate field value entered","field":{"name":"Theater 1"}}
```

3. Passos para Reprodução

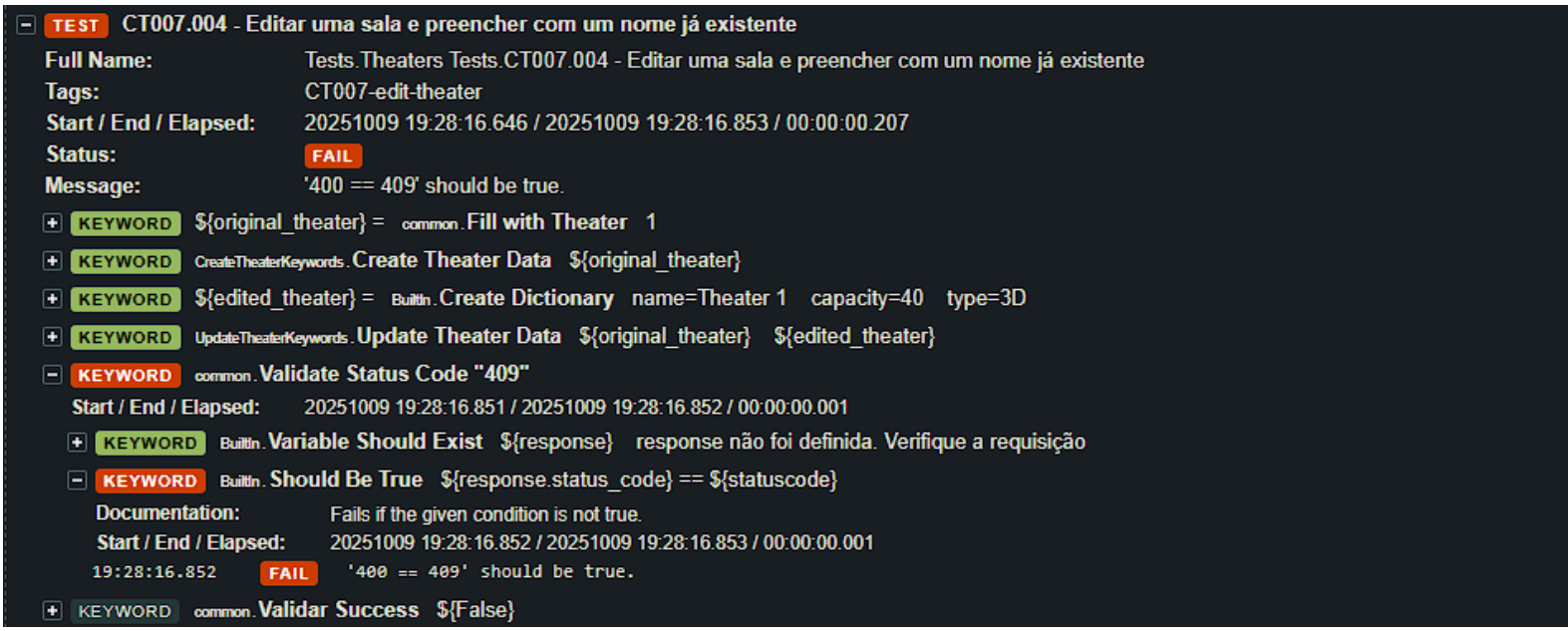
Liste os passos exatos para reproduzir o defeito, incluindo:

[Passo 1] Enviar requisição PUT para `/theaters` com os seguintes campos:

```
"name": "string",
"capacity": 1,
"type": "standard"}
```

[Passo 2] Observar o Status Code retornado

4. Evidências



5. Classificação da Issue

- **Prioridade:** Baixa
- **Gravidade/Impacto:** Baixa
- **Frequência:** Ocorre consistentemente
- **Tipo de Defeito:** Erro técnico / implementação