Cenários de Teste (Backend)

CT-001 – Registro de novo usuário com dados válidos

Prioridade: 🔴 Alta
Tags: @api @registro @positivo @regressivo
Isolamento de Dados: E-mail com timestamp deve ser usado para evitar duplicação

Pré-condição:

- API em execução
- Banco de dados acessível

Dados de Teste:

```
{
  "name": "Hannah QA",
  "email": "hannah_qa_<timestamp>@example.com",
  "password": "Teste@123"
}
```

Passos Detalhados:

1. Abrir Postman ou script de automação
2. Selecionar método POST
3. Enviar para `/auth/register`
4. Inserir o body com os dados acima
5. Executar a requisição

Resultado Esperado:

- `201 Created`
- JSON com objeto `data` contendo: `id`, `name`, `email`, `role`, `token`

Pós-condição:

- Usuário criado na base
- Token vinculado à conta

CT-002 – Tentativa de registro com e-mail duplicado

Prioridade: 🔴 Alta
Tags: @api @registro @validação @negativo @regressivo
Isolamento de Dados: Executar antes o CT-001 ou cadastrar usuário com e-mail `"hannah_qa_<timestamp>@example.com"`

Pré-condição:

- E-mail `“hannah_qa_<timestamp>@example.com"` já cadastrado

Dados de Teste:

```
}
  "name": "Hannah QA",
  "email": "hannah_qa_<timestamp>@example.com",
  "password": "Teste@1234"
}
```

Passos Detalhados:

1. Abrir Postman ou script de automação
2. Selecionar método POST
3. Enviar request para `http://localhost:3000/api/v1/auth/register`
4. Inserir o body com os dados acima
5. Executar a requisição

Resultado Esperado:

- `400 Bad Request`
- Mensagem: `"User already exists"`

Pós-condição:

- Nenhum novo usuário deve ser criado

CT-003 – Registro com e-mail inválido

Prioridade: 🟡 Média
Tags: @api @registro @validação @negativo @regressivo
Isolamento de Dados: Desnecessário (valor inválido proposital)

Pré-condição:

- API e banco funcionando

Dados de Teste:

```
{
  "name": "Teste Malformado",
  "email": "hannah_qa_<timestamp>example.com",
  "password": "Teste@123"
}
```

Passos Detalhados:

1. Abrir Postman ou script de automação
2. Enviar POST para `/auth/register`
3. Inserir o body com dados acima
4. Executar a requisição

Resultado Esperado:

- `400 Bad Request`
- Mensagem: `"Invalid email format"`

Pós-condição:

- Nenhum dado inserido no banco

CT-004 – Registro com senha fraca (sem caracteres especiais)

Prioridade: 🟡 Média
Tags: @api @registro @validação @negativo @regressivo
Isolamento de Dados: E-mail com timestamp único

Pré-condição:

- API em execução e conectada ao banco

Dados de Teste:

```
{
  "name": "Senha Fraca",
  "email": "senha_fraca_<timestamp>@example.com",
  "password": "12345678"
}
```

Passos Detalhados:

1. Abrir Postman ou script de automação
2. Enviar request POST para `/auth/register`
3. Inserir o body com os dados acima
4. Executar a requisição

Resultado Esperado:

- `400 Bad Request`
- Mensagem de erro indicando política de senha inválida

Pós-condição:

- Nenhum usuário criado

Observações:

- Este teste assume que há política de senha forte implementada

---

CT-005 – Login com credenciais válidas

Prioridade: 🔴 Crítica
Tags: @api @login @autenticação @positivo @regressivo
Isolamento de Dados: Usuário deve ser previamente criado com o e-mail de teste

Pré-condição:

- Usuário existente com e-mail e senha válidos

Dados de Teste:

```
{
  "email": "hannah_login@example.com",
  "password": "Teste@123"
```

```
}
```

Passos Detalhados:

1. Abrir Postman ou script de automação
2. Enviar POST para `auth/login`
3. Inserir o body com os dados acima
4. Executar a requisição

Resultado Esperado:

- `200 OK`
- Token JWT retornado no campo `token`

Pós-condição:

- Token pode ser armazenado para testes seguintes

CT-006 – Tentativa de login com senha inválida

Prioridade: 🔴 Crítica
Tags: @api @login @validação @negativo @regressivo
Isolamento de Dados: Usuário com e-mail válido deve existir previamente

Pré-condição:

- Usuário existente com e-mail correto, senha errada

Dados de Teste:

```
{
  "email": "hannah_login@example.com",
  "password": "senhaIncorreta"
}
```

Passos Detalhados:

1. Abrir Postman ou script de automação
2. Enviar POST para `/auth/login`
3. Inserir o body com dados acima
4. Executar a requisição

Resultado Esperado:

- `400 Bad Request`
- Mensagem: `"Invalid credentials"`

Pós-condição:

- Nenhum token emitido

CT-007 – Consulta ao perfil com token válido

Prioridade: 🔴 Alta
Tags: @api @perfil @autenticação @positivo @regressivo

Isolamento de Dados: Token válido obtido via login

Pré-condição:

- Usuário autenticado

Dados de Teste:
Header: `Authorization: Bearer <token-válido>`

Passos Detalhados:

1. Abrir Postman ou script de automação
2. Enviar GET para `/auth/me`
3. Inserir Header de autorização com token válido
4. Executar a requisição

Resultado Esperado:

- `200 OK`
- JSON com dados do usuário autenticado

Pós-condição:

- Nenhuma alteração no estado do sistema

CT-008 – Consulta ao perfil SEM token

Prioridade: 🔴 Crítica
Tags: @api @perfil @segurança @negativo @regressivo
Isolamento de Dados: Não aplicável

Pré-condição:

- Nenhum token fornecido

Passos Detalhados:

1. Abrir Postman ou script de automação
2. Enviar GET para `/auth/me`
3. Não inserir Header de autorização
4. Executar a requisição

Resultado Esperado:

- `401 Unauthorized`
- Mensagem: `"Not authorized, no token"`

Pós-condição:

- Nenhum dado retornado

CT-009 – Consulta ao perfil com token inválido

Prioridade: 🔴 Crítica
Tags: @api @perfil @segurança @negativo @regressivo
Isolamento de Dados: Token inválido pode ser fixo

Dados de Teste:
Header: `Authorization: Bearer abc.invalid.token`

Passos Detalhados:

1. Abrir Postman ou script de automação
2. Enviar GET para `/auth/me`
3. Inserir token inválido no Header
4. Executar a requisição

Resultado Esperado:

- `403 Forbidden`
- Mensagem: `"Not authorized, invalid token"`

CT-010 – Atualização de nome no perfil com dados válidos

Prioridade: 🟡 Média
Tags: @api @perfil @atualização @positivo @regressivo
Isolamento de Dados: A ação atualiza o dado permanentemente, considerar reset

Pré-condição:

- Usuário autenticado com token válido

Dados de Teste:

```
{
  "name": "Hannah QA Atualizada",
  "currentPassword": "Teste@123"
}
```

Passos Detalhados:

1. Abrir Postman ou script de automação
2. Enviar PUT para `/auth/profile`
3. Inserir Header com token
4. Inserir o body com os dados acima
5. Executar a requisição

Resultado Esperado:

- `200 OK`
- Campo `name` atualizado no banco

CT-011 – Listar todos os filmes

Prioridade: 🔴 Alta
Tags: @api @filmes @consulta @positivo @regressivo
Isolamento de Dados: Base de dados deve conter ao menos um filme válido

Pré-condição:

- API em execução

- Banco populado com filmes

Dados de Teste:

- Nenhum

Passos Detalhados:

1. Abrir Postman ou script de automação
2. Enviar GET para `/movies`
3. Executar a requisição

Resultado Esperado:

- `200 OK`
- Lista de filmes com: `title`, `duration`, `classification`, `genres`, `releaseDate`

Pós-condição:

- Nenhuma alteração de estado

CT-012 – Obter detalhes de um filme válido

Prioridade: 🔴 Alta
Tags: @api @filmes @consulta @positivo @regressivo
Isolamento de Dados: Garantir existência de um filme válido criado previamente

Pré-condição:

- Filme cadastrado com ID conhecido

Dados de Teste:

- ID do filme existente

Passos Detalhados:

1. Abrir Postman ou script de automação
2. Enviar GET para `movies/{id}`
3. Substituir `{id}` pelo ID do filme existente
4. Executar a requisição

Resultado Esperado:

- `200 OK`
- JSON com todos os campos do filme

Pós-condição:

- Nenhuma alteração no banco

CT-013 – Obter detalhes com ID inválido

Prioridade: 🔴 Alta
Tags: @api @filmes @validação @negativo @regressivo
Isolamento de Dados: Não aplicável (valor inválido proposital)

Pré-condição:

- Nenhuma

Dados de Teste:

- ID: `"abc123"`

Passos Detalhados:

1. Abrir Postman ou script de automação
2. Enviar GET para `movies/abc123`
3. Executar a requisição

Resultado Esperado:

- `400 Bad Request`
- Mensagem: `"Invalid movie ID format or movie not found"`

Pós-condição:

- Nenhum dado exposto

CT-014 – Criar um novo filme com dados válidos

Prioridade: 🔴 Alta
Tags: @api @filmes @criação @positivo @admin
Isolamento de Dados: Usar título único ou limpar base após execução

Pré-condição:

- Usuário autenticado como ADMIN

Dados de Teste:

```json
{
  "title": "O Jogo da Imitação",
  "synopsis": "A história de Alan Turing e a quebra dos códigos nazistas.",
  "director": "Morten Tyldum",
  "genres": ["Drama", "Biografia"],
  "duration": 113,
  "classification": "PG-13",
  "poster": "imitationgame.jpg",
  "releaseDate": "2014-11-28"
}
```

Passos Detalhados:

1. Abrir Postman ou script de automação
2. Enviar POST para `http://localhost:3000/api/v1/movies`
3. Header: `Authorization: Bearer <token-admin>`
4. Inserir body com dados acima
5. Executar a requisição

Resultado Esperado:

- `201 Created`

- Filme criado e refletido na resposta

Pós-condição:

- Filme inserido na base

CT-015 – Tentativa de criação de filme sem token

Prioridade: 🔴 Crítica
Tags: @api @filmes @segurança @negativo @admin
Isolamento de Dados: Usar mesmo payload do CT-014

Pré-condição:

- Nenhum token fornecido

Passos Detalhados:

1. Abrir Postman ou script de automação
2. Enviar POST para `/movies`
3. Sem header de Authorization
4. Body com dados válidos
5. Executar requisição

Resultado Esperado:

- `401 Unauthorized`
- Mensagem: `"Not authorized"`

Pós-condição:

- Nenhum filme inserido

CT-016 – Atualizar dados de um filme existente (admin)

Prioridade: 🟡 Média
Tags: @api @filmes @atualização @positivo @admin
Isolamento de Dados: Filme previamente criado

Pré-condição:

- Filme existente com ID válido
- Token de administrador válido

Dados de Teste: `"duration": 120`

Passos Detalhados:

1. Abrir Postman ou script de automação
2. Enviar PUT para `/movies/{id}`
3. Inserir Authorization: Bearer `<token-admin>`
4. Inserir body com campo a ser alterado
5. Executar requisição

Resultado Esperado:

- `200 OK`
- Campos alterados refletidos na resposta

CT-017 – Excluir um filme existente com credenciais válidas

Prioridade: 🔴 Alta
Tags: @api @filmes @deleção @positivo @admin
Isolamento de Dados: Filme deve ter sido criado previamente

Pré-condição:

- Filme criado com ID conhecido
- Usuário ADMIN autenticado

Dados de Teste:

- movieId: `"60d0fe4f5311236168a109cb"`

Passos Detalhados:

1. Abrir Postman ou script de automação
2. Enviar DELETE para `/movies/{id}`
3. Substituir `{id}` pelo movieId
4. Header: Authorization com token admin
5. Executar requisição

Resultado Esperado:

- `200 OK`
- Mensagem: `"Movie removed"`

CT-018 – Criando uma sessão com dados válidos

Prioridade: 🔴 Alta
Tags: @api @sessões @consulta @positivo @admin
Isolamento de Dados:

Pré-condição:

Dados de Teste:


```
{
    "movie": {
        "_id": "685af8b2dd66e8849ea14e0f",
        "title": "O Jogo da Imitação",
        "duration": 113,
        "poster": "imitationgame.jpg",
        "id": "685af8b2dd66e8849ea14e0f"
    },
    "theater": {
        "_id": "685af8b2dd66e8849ea14e14",
        "name": "Theater QA",
        "type": "IMAX",
        "id": "685af8b2dd66e8849ea14e14"
    },
```

```
  "datetime": "2025-06-28T21:08:23.739Z",
  "fullPrice": 20,
  "halfPrice": 10
  "seats": [
        {
            "row": "A",
            "number": 1,
            "status": "available"
        },
        ...
        {
            "row": "A",
            "number": 10,
            "status": "available"
        },
        {
            "row": "B",
            "number": 11,
            "status": "available"
        },
        ...
        {
            "row": "B",
            "number": 20,
            "status": "available"
        },
        {
            "row": "C",
            "number": 21,
            "status": "available"
        },
       ...
        {
            "row": "C",
            "number": 30,
            "status": "available"
        },
        {
            "row": "D",
            "number": 31,
            "status": "available"
        },
        ...
        {
            "row": "D",
            "number": 35,
            "status": "available"
        }
    ]
}
```

Passos Detalhados:

1. Abrir Postman ou script de automação
2. Enviar POST para `/sessions`
3. Inserir Authorization: Bearer `<token-admin>`
4. Inserir body com dados acima

5. Executar requisição

Resultado Esperado:

- `201 Created`
- Sessão criada e refletida na resposta

Pós-condição:

- Sessão inserida na base

CT-019 – Listar sessões com dados válidos

Prioridade: 🔴 Alta
Tags: @api @sessões @criação @positivo @regressivo
Isolamento de Dados: Sessões precisam estar previamente criadas

Pré-condição:

- Sessões válidas cadastradas

Passos Detalhados:

1. Abrir Postman ou script de automação
2. Enviar GET para `/sessions`
3. Executar requisição

Resultado Esperado:

- `201 Created`
- Lista com dados de todas as sessões criadas.

CT-020 – Consultar assentos ocupados por ID

Prioridade: 🔴 Alta
Tags: @api @sessões @consulta @validação
Isolamento de Dados: Reserva já criada anteriormente

Pré-condição:

- Assentos reservados na sessão

Passos Detalhados:

1. Abrir Postman ou script de automação
2. Enviar GET para `/sessions/{id}`
3. Verificar status dos assentos
4. Executar requisição

Resultado Esperado:

- `200 OK`
- Assentos marcados como `"reserved"`

CT-021 – Resetando status dos assentos de uma sessão válida

Prioridade: 🔴 Alta
Tags: @api @sessões @atualização @positivo @admin
Isolamento de Dados: Sessão já criada anteriormente

Pré-condição:

Pré-condição:

- Sessão criada com ID válido
- Usuário ADMIN autenticado

Passos Detalhados:

1. Abrir Postman ou script de automação
2. Enviar PUT para `/sessions/{id}/reset-seats`
3. Inserir Authorization: Bearer `<token-admin>`
4. Executar requisição

Resultado Esperado:

- `200 OK`
- Mensagem: `"All seats reset to available"`

CT-022 – Exclusão de sessão com dados válidos

Prioridade: 🔴 Alta
Tags: @api @sessões @deleção @positivo @admin
Isolamento de Dados: Sessão deve ter sido criado previamente

Pré-condição:

- Sessão criada com ID válido
- Usuário ADMIN autenticado

Dados de Teste:

- sessionID: `""`

Passos Detalhados:

1. Abrir Postman ou script de automação
2. Enviar DELETE para `sessions/{id}`
3. Substituir `{id}` pelo sessionId
4. Inserir Header: Authorization com token admin
5. Executar requisição

Resultado Esperado:

- `200 OK`
- Mensagem: `"Session deleted successfully"`

CT-023 – Seleção de múltiplos assentos disponíveis

Prioridade: 🔴 Crítica
Tags: @api @reservas @criação @positivo
Isolamento de Dados: Sessão com assentos livres deve ser configurada

Pré-condição:

- Sessão ativa e com assentos disponíveis
- Usuário autenticado

Dados de Teste:

```
{
  "session": "<session_id>",
  "seats": [
    {"row": "C", "number": 21, "type": "full"},
    {"row": "C", "number": 22, "type": "half"}
  ],
  "paymentMethod": "credit_card"
}
```

Passos Detalhados:

1. Abrir Postman ou script de automação
2. Enviar POST para `/reservations`
3. Header: Authorization com token do usuário
4. Inserir body com dados acima
5. Executar requisição

Resultado Esperado:

- `201 Created`
- Reserva registrada com assentos associados

CT-024 – Tentativa de selecionar assentos já reservados

Prioridade: 🔴 Alta
Tags: @api @reservas @validação @negativo
Isolamento de Dados: Reutilizar assentos já reservados no CT-019

Pré-condição:

- Assentos já reservados na sessão

Dados de Teste:

```
{
  "session": "<session_id>",
  "seats": [
    {"row": "C", "number": 21, "type": "full"},
    {"row": "C", "number": 22, "type": "half"}
  ],
  "paymentMethod": "credit_card"
}
```

Passos Detalhados:

1. Abrir Postman ou script de automação
2. Enviar POST para `/reservations`
3. Inserir body com os mesmos assentos do CT-019
4. Executar requisição

Resultado Esperado:

- `400 Bad Request`
- Mensagem sobre indisponibilidade dos assentos (occupied)

CT-025 – Excluindo uma reserva com dados válidos

Prioridade: 🔴 Alta
Tags: @api @reservas @deleção @positivo
Isolamento de Dados: Sessão ativa com assentos já reservados pelo usuário.

Pré-condição:

- ID de uma sessão válida e ativa
- Assentos reservados previamente
- Usuário autenticado

Passos Detalhados:

1. Abrir Postman ou script de automação
2. Enviar POST para `/reservations/{id}`
3. Header: Authorization com token do usuário
4. Executar requisição

Resultado Esperado:

- `200 OK`
- Mensagem: `“Reservation deleted successfully”`

CT-026 – Listagem de usuários com token de admin

Prioridade: 🟡 Média

Tags: @consulta @users @admin-only @positive
Pré-condição:

- API em execução
- Usuário autenticado com role **admin**
- Token JWT válido

Dados de Teste:

- Usuário ADMIN autenticado

Passos Detalhados:

1. Abrir o Postman ou script de automação
2. Selecionar método GET para `/users`

3. Inserir Header: Authorization com token admin
4. Adicionar parâmetros de paginação (opcional)
5. Executar a requisição

Resultado Esperado:

- `200 OK`
- Lista de usuários com metadados de paginação

Pós-condição:

- Nenhuma alteração no estado do sistema

CT-027 – Listagem de usuários sem token

Prioridade: 🔴 Crítica

Tags: @users @auth @negative @security
Pré-condição:

- API em execução

Passos Detalhados:

1. Enviar GET para `/users` sem header de autorização
2. Executar a requisição

Resultado Esperado:

- `401 Unauthorized`
- Mensagem: "Not authorized"

Pós-condição:

- Nenhum dado exposto

CT-028 – Exclusão de usuário com token de admin

Prioridade: 🔴 Alta
Tags: @api @users @deleção @admin
Isolamento de Dados: Usuário criado apenas para este teste

Pré-condição:

- ID do usuário existente
- Token admin válido

Passos Detalhados:

1. Enviar DELETE para `/users/{id}` com header de autorização
2. Substituir `{id}` por ID válido
3. Executar requisição

Resultado Esperado:

- `200 OK`

- Mensagem: `"User deleted successfully"`

CT-029 – Exclusão de usuário com reservas ativas

Prioridade: 🟡 Média
Tags: @api @users @validação @negativo
Isolamento de Dados: Criar reservas associadas ao usuário antes do teste

Pré-condição:

- Usuário com reservas em aberto

Passos Detalhados:

1. Enviar DELETE para `/users/{id}` com token admin
2. Inserir ID de usuário com reservas
3. Executar requisição

Resultado Esperado:

- `409 Conflict`
- Mensagem: `"Cannot delete user with active reservations"`

CT-030 – Criação de um novo cinema com dados válidos (admin)

Prioridade: 🔴 Alta
Tags: @api @theaters @criação @positivo @admin
Isolamento de Dados: Nome do cinema deve ser único

Pré-condição:

- Usuário autenticado como ADMIN
- Token válido no Header

Dados de Teste:

```
{
  "name": "Cine QA Plaza",
  "capacity": 150,
  "type": "IMAX"
}
```

Passos Detalhados:

1. Enviar `POST` para: `/theaters`
2. Incluir o Header: `Authorization: Bearer <token-admin>`
3. Adicionar o body com os dados de teste
4. Executar a requisição

Resultado Esperado:

- `201 Created`
- JSON com os dados do cinema recém-criado

Pós-condição:

- Cinema salvo na base

CT-031 – Criação de cinema com campo obrigatório ausente (admin)

Prioridade: 🟡 Média
Tags: @api @theaters @validação @negativo
Isolamento de Dados: Campo ausente proposital

Pré-condição:
Usuário autenticado como admin

Dados de Teste:

```
{
  "name": "Cine Plaza",
  "capacity": 100,
  "type": "standard"
}
```

Passos Detalhados:

1. Enviar POST para `/theaters`
2. Header com token admin
3. Inserir body como acima.
4. Executar requisição

Resultado Esperado:

- 400 Bad Request
- Mensagem: `"Theater name is required"`

CT-032 – Atualizar dados de um cinema existente (admin)

Prioridade: 🟡 Média
Tags: @api @theaters @atualização @positivo @admin
Isolamento de Dados: O cinema deve ter sido criado previamente

Pré-condição:

- ID de cinema existente
- Usuário autenticado como ADMIN

Dados de Teste:

```
{
  "name": "Cine QA Plaza Atualizado",
  "capacity": 200,
  "type": "standard"
}
```

Passos Detalhados:

1. Enviar `PUT` para: `/theaters/{id}`
2. Incluir o Header: `Authorization: Bearer <token-admin>`

3. Inserir o body com os dados atualizados
4. Executar a requisição

Resultado Esperado:

- `200 OK`
- JSON refletindo os novos dados do cinema

Pós-condição:

- Dados atualizados no banco.

CT-033 – Listagem de todos os cinemas (theaters)

Prioridade: 🔴 Alta
Tags: @api @theaters @consulta @positivo
Isolamento de Dados: Banco deve conter ao menos um cinema válido previamente criado

Pré-condição:

- API em execução
- Banco de dados com cinemas válidos

Passos Detalhados:

1. Abrir Postman ou script de automação
2. Enviar requisição GET para: `/theaters`
3. Executar a requisição

Resultado Esperado:

- `200 OK`
- JSON com array de objetos contendo: `_id`, `name`, `capacity`, `type`, `createdAt`

Pós-condição:

- Nenhuma alteração no estado do sistema

CT-034 – Exclusão de um cinema válido (admin)

Prioridade: 🔴 Alta
Tags: @api @theaters @deleção @positivo @admin
Isolamento de Dados: O cinema não pode estar vinculado a sessões ativas

Pré-condição:

- Cinema existente com ID conhecido
- Usuário autenticado como ADMIN

Passos Detalhados:

1. Enviar DELETE para: `/theaters/{id}`
2. Incluir o Header: `Authorization: Bearer <token-admin>`
3. Substituir `{id}` pelo ID real do cinema
4. Executar a requisição

Resultado Esperado:

- `200 OK`
- Mensagem: `"Theater deleted successfully"`

Pós-condição:

- Cinema removido da base.

CT-035 – Exclusão de cinema vinculado a sessões ativas

Prioridade: 🔴 Alta
Tags: @api @theaters @validação @negativo @admin
Isolamento de Dados: Associar sessões ao cinema previamente

Pré-condição:
Cinema com sessões vinculadas

Passos Detalhados:

1. Enviar DELETE para `/theaters/{id}`
2. Inserir token admin válido
3. Executar requisição

Resultado Esperado:

- `409 Conflict`
- Mensagem: `"Cannot delete theater with active sessions"`.
