

Relatório de Teste - Sistema de Reservas de Salas de Reunião

Projeto: Sistema de Reservas de Salas de Reunião

Descrição: Sistema para reservas de salas de reunião com frontend em ReactJS, backend em Laravel, e banco de dados PostgreSQL.

1. Objetivo do Teste

Garantir que o sistema de reservas atenda a todos os requisitos de funcionalidade, qualidade de código, UX e conformidade com os princípios RESTful. Validar que o sistema é capaz de lidar com as operações CRUD para reservas e exibe as informações corretamente no frontend.

2. Escopo do Teste

- Testes de backend (API Laravel)
- Testes de frontend (interface React)
- Testes de banco de dados (PostgreSQL)
- Testes de integração entre frontend e backend
- Validação de requisitos de negócio

3. Ambiente de Teste

- **Backend:** PHP 8.x, Laravel 10.48.22, PostgreSQL 13+ → 14.13
- **Frontend:** ReactJS → `"react": "^18.3.1", "react-dom": "^18.3.1"`
- , HTML5, CSS3
- `package.json`{

```
"name": "reserva-salas-frontend",
"version": "0.1.0",
"private": true,
"dependencies": {
"@testing-library/jest-dom": "^5.17.0",
"@testing-library/react": "^13.4.0",
"@testing-library/user-event": "^13.5.0",
"axios": "^1.7.7",
"bootstrap": "^5.3.3",
"date-fns": "^4.1.0",
"font-awesome": "^4.7.0",
"react": "^18.3.1",
"react-bootstrap": "^2.10.5",
"react-dom": "^18.3.1",
"react-router-dom": "^6.27.0",
"react-scripts": "5.0.1",
"react-table": "^7.8.0",
"web-vitals": "^2.1.4"
},
"scripts": {
"start": "react-scripts start",
"build": "react-scripts build",
```

```

"test": "react-scripts test",
"eject": "react-scripts eject"
},
"eslintConfig": {
  "extends": [
    "react-app",
    "react-app/jest"
  ],
},
"browserslist": {
  "production": [
    ">0.2%",
    "not dead",
    "not op_mini all"
  ],
  "development": [
    "last 1 chrome version",
    "last 1 firefox version",
    "last 1 safari version"
  ],
},
"devDependencies": {
  "@babel/plugin-proposal-private-property-in-object": "^7.21.11"
}
}

```

- **Ambiente local:** Node.js v20.18.0, Composer → Composer version 2.8.1
- PHP version 8.1.30 (/usr/bin/php8.1)
- **Ferramentas de teste:** Postman

4. Casos de Teste

Backend - Laravel API (CRUD de Reservas) TESTE MANUAL

ID	Caso de Teste	Procedimento	Resultado Esperado	Status
1	Criar reserva válida	Enviar requisição POST para <code>/api/reservas</code> com dados válidos da reserva.	Reserva criada com sucesso, resposta HTTP 201 com os dados da reserva	OK
2	Criar reserva em horário duplicado	Enviar requisição POST para <code>/api/reservas</code> em um horário já reservado para a mesma sala.	Erro de validação, resposta HTTP 400 com mensagem de conflito de horário	OK
3	Criar reserva para data passada	Enviar requisição POST para <code>/api/reservas</code> com uma data/hora de início anterior à data atual.	Erro de validação, resposta HTTP 400 com mensagem de erro de data inválida	OK
4	Editar reserva existente	Enviar requisição PUT para <code>/api/reservas/{id}</code> com novos dados para uma reserva existente.	Reserva atualizada, resposta HTTP 200 com dados atualizados	OK
5	Cancelar reserva	Enviar requisição DELETE para <code>/api/reservas/{id}</code> para cancelar uma reserva existente.	Reserva cancelada, resposta HTTP 200 com confirmação de cancelamento	OK
6	Visualizar todas as reservas	Enviar requisição GET para <code>/api/reservas</code> .	Lista de reservas exibida, resposta HTTP 200 com dados de todas as reservas	OK
7	Visualizar reserva específica	Enviar requisição GET para <code>/api/reservas/{id}</code> para uma reserva específica.	Dados da reserva exibidos, resposta HTTP 200 com informações detalhadas da reserva	OK

Relatório de APIs - Reservas de Salas

<https://bold-comet-4803.postman.co/workspace/My-Workspace~c718c4e0-47a9-4e35-8d62-6ff68ad19285/collection/9327314-7b1f9ca1-3a09-4359-8284-8377f0509815?action=share&creator=9327314>

CRUD Salas

Reservas

1. Listar Reservas

- **Método:** GET
 - **Endpoint:** <http://localhost:8000/api/reservas>
 - **Descrição:** Lista todas as reservas.
-

2. Criar Reserva

- **Método:** POST
- **Endpoint:** <http://localhost:8000/api/reservas>
- **Descrição:** Cria uma nova reserva.

Request Headers

- Accept: application/json

Body (raw - JSON)

```
{  
  "sala_id": 1,  
  "responsavel": "Teste DE",  
  "inicio": "2024-10-21T10:00:00",  
  "fim": "2024-10-21T12:00:00"  
}
```

3. Visualizar uma Reserva

- **Método:** GET
- **Endpoint:** <http://localhost:8000/api/reservas/18>
- **Descrição:** Visualiza os detalhes de uma reserva específica.

4. Atualizar Reserva

- **Método:** PUT
- **Endpoint:** <http://localhost:8000/api/reservas/18>
- **Descrição:** Atualiza uma reserva existente.

Body (raw - JSON)

```
{  
  "sala_id": 1,  
  "responsavel": "Teste DE atualizar",  
  "inicio": "2024-10-21T10:00:00",  
  "fim": "2024-10-21T12:00:00"  
}
```

5. Deletar uma Reserva

- **Método:** DELETE
- **Endpoint:** <http://localhost:8000/api/reservas/18>
- **Descrição:** Deleta uma reserva específica.

6. Verificar Disponibilidade da Reserva

- **Método:** POST
- **Endpoint:** <http://localhost:8000/api/reservas/verificar-disponibilidade>
- **Descrição:** Verifica a disponibilidade de uma sala para uma determinada faixa de horários.

Request Headers

- **Accept:** application/json

Body (raw - JSON)

```
{  
  "sala_id": 1,  
  "inicio": "2024-10-26 10:00:00",  
  "fim": "2024-10-26 12:00:00"  
}
```

CRUD Salas

Salas

1. Listar Salas

- **Método:** GET
- **Endpoint:** <http://localhost:8000/api/salas>
- **Descrição:** Lista todas as salas.

Aqui está o relatório formatado para colar no Word, sobre as APIs de reservas e salas:

Relatório de APIs - Reservas de Salas

CRUD Salas

Reservas

1. Listar Reservas

- **Método:** GET
 - **Endpoint:** <http://localhost:8000/api/reservas>
 - **Descrição:** Lista todas as reservas.
-

2. Criar Reserva

- **Método:** POST
- **Endpoint:** <http://localhost:8000/api/reservas>
- **Descrição:** Cria uma nova reserva.

Request Headers

- Accept: application/json

Body (raw - JSON)

```
json
Copiar código
{
  "sala_id": 1,
  "responsavel": "Teste DE",
  "inicio": "2024-10-21T10:00:00",
  "fim": "2024-10-21T12:00:00"
}
```

3. Visualizar uma Reserva

- **Método:** GET
 - **Endpoint:** <http://localhost:8000/api/reservas/18>
 - **Descrição:** Visualiza os detalhes de uma reserva específica.
-

4. Atualizar Reserva

- **Método:** PUT
- **Endpoint:** <http://localhost:8000/api/reservas/18>
- **Descrição:** Atualiza uma reserva existente.

Body (raw - JSON)

```
json
Copiar código
{
  "sala_id": 1,
  "responsavel": "Teste DE atualizar",
  "inicio": "2024-10-21T10:00:00",
  "fim": "2024-10-21T12:00:00"
}
```

5. Deletar uma Reserva

- **Método:** DELETE
 - **Endpoint:** <http://localhost:8000/api/reservas/18>
 - **Descrição:** Deleta uma reserva específica.
-

6. Verificar Disponibilidade da Reserva

- **Método:** POST
- **Endpoint:** <http://localhost:8000/api/reservas/verificar-disponibilidade>
- **Descrição:** Verifica a disponibilidade de uma sala para uma determinada faixa de horários.

Request Headers

- Accept: application/json

Body (raw - JSON)

```
json
Copiar código
{
  "sala_id": 1,
  "inicio": "2024-10-26 10:00:00",
  "fim": "2024-10-26 12:00:00"
}
```

CRUD Salas

Salas

1. Listar Salas

- **Método:** GET
 - **Endpoint:** <http://localhost:8000/api/salas>
 - **Descrição:** Lista todas as salas.
-

2. Criar uma Sala

- **Método:** POST
- **Endpoint:** <http://localhost:8000/api/salas>
- **Descrição:** Cria uma nova sala.

Request Headers

- Accept: application/json

Body (raw - JSON)

```
{  
  "nome": "Sala AAAAAAA",  
  "capacidade": 20,  
  "numero": 101  
}
```

3. Visualizar uma Sala

- **Método:** GET
- **Endpoint:** <http://localhost:8000/api/salas/3>
- **Descrição:** Visualiza os detalhes de uma sala específica.

4. Atualizar uma Sala

- **Método:** PUT
- **Endpoint:** <http://localhost:8000/api/salas/3>
- **Descrição:** Atualiza os detalhes de uma sala existente.

Request Headers

- Accept: application/json

Body (raw - JSON)

```
{  
  "nome": "Sala A Atualizada",  
  "capacidade": 25,  
  "numero": 102  
}
```

5. Deletar uma Sala

- **Método:** DELETE
- **Endpoint:** <http://localhost:8000/api/salas/6>
- **Descrição:** Deleta uma sala específica.

Request Headers

- Accept: application/json

Banco de Dados – PostgreSQL - NOME: reserva_salas_db, Senha: root

Configuração da conexão "reserva_salas_db"

Configurações de conexão

Configuração de conexão para PostgreSQL

Configurações de conexão

Inicialização

Comandos de shell

Identificação de cliente

Transações

Geral

Metadados

Erros e timeouts

Transferência de dados

> Editor de dados

> Editor SQL

Principal PostgreSQL Propriedades do driver SSH SSL + Configurações de rede...

Servidor

Conecte usando: ☒ Host ☐ URL

URL: jdbc:postgresql://localhost:5432/reserva_salas_db

Host: localhost Porta: 5432

Banco de dados: reserva_salas_db ☐ Exibir todos os bancos de dados

Autenticação

Autenticação: Database Native

Nome de usuário: root

Senha: **** ☒ Salvar senha

Avançado

Role da sessão: Cliente local: /usr/lib/postgresql/14

[Informação de variáveis de conexão](#) [Documentação de banco de dados](#)

Nome do driver: PostgreSQL Configurações de driver Licença do driver

Testar conexão ... Cancelar Ok

.env

APP_NAME=Laravel

APP_ENV=local

APP_KEY=base64:+U1x2taVadFor/+IbSxPIVr3U4qCySptYEGjPkD7yQQ=

APP_DEBUG=true

APP_URL=http://localhost

LOG_CHANNEL=stack

LOG_DEPRECATIONS_CHANNEL=null

LOG_LEVEL=debug

DB_CONNECTION=pgsql

DB_HOST=127.0.0.1

DB_PORT=5432

DB_DATABASE=reserva_salas_db

DB_USERNAME=root

DB_PASSWORD=root

BROADCAST_DRIVER=log

CACHE_DRIVER=file
FILESYSTEM_DISK=local
QUEUE_CONNECTION=sync
SESSION_DRIVER=file
SESSION_LIFETIME=120

MEMCACHED_HOST=127.0.0.1

REDIS_HOST=127.0.0.1
REDIS_PASSWORD=null
REDIS_PORT=6379

MAIL_MAILER=smtp
MAIL_HOST=mailpit
MAIL_PORT=1025
MAIL_USERNAME=null
MAIL_PASSWORD=null
MAIL_ENCRYPTION=null
MAIL_FROM_ADDRESS="hello@example.com"
MAIL_FROM_NAME="\${APP_NAME}"

AWS_ACCESS_KEY_ID=
AWS_SECRET_ACCESS_KEY=
AWS_DEFAULT_REGION=us-east-1
AWS_BUCKET=
AWS_USE_PATH_STYLE_ENDPOINT=false

PUSHER_APP_ID=
PUSHER_APP_KEY=
PUSHER_APP_SECRET=
PUSHER_HOST=
PUSHER_PORT=443
PUSHER_SCHEME=https
PUSHER_APP_CLUSTER=mt1

VITE_APP_NAME="\${APP_NAME}"
VITE_PUSHER_APP_KEY="\${PUSHER_APP_KEY}"
VITE_PUSHER_HOST="\${PUSHER_HOST}"
VITE_PUSHER_PORT="\${PUSHER_PORT}"
VITE_PUSHER_SCHEME="\${PUSHER_SCHEME}"
VITE_PUSHER_APP_CLUSTER="\${PUSHER_APP_CLUSTER}"

ID	Caso de Teste	Procedimento	Resultado Esperado	Status
1	Criar tabelas com migrations	Executar <code>php artisan migrate</code> para criar tabelas no PostgreSQL.	Tabelas criadas corretamente com estrutura definida (salas e reservas). <code>reserva-salas-backend/backend\$ php artisan migrate</code>	OK
2	Inserir dados de teste	Inserir registros de teste para salas e reservas.	Dados inseridos com sucesso, verificáveis por meio de comandos SQL, <code>php artisan db:seed --class=SalaSeeder</code> , <code>php artisan db:seed --class=ReservaSeeder</code>	OK
3	Testar constraint de horário	Inserir uma reserva para uma sala já reservada no mesmo horário.	Violação de constraint, impedindo inserção de reserva conflitante.	OK
4	Testar A data fim deve ser posterior à data início.	Inserir uma reserva onde a data fim é posterior a data início	Violação de constraint, impedindo inserção de reserva	OK
4	Excluir registros	Excluir uma reserva específica.	Registro excluído com sucesso, sem afetar outros registros.	OK

FRONTEND

Frontend - ReactJS (Interface do Usuário)

ID	Caso de Teste	Procedimento	Resultado Esperado	Status
1	Visualizar todas as salas	Acessar a interface principal e verificar se todas as salas e suas disponibilidades são exibidas.	Todas as salas disponíveis exibidas com status de ocupação/reserva atualizado	OK
2	Realizar reserva	Preencher formulário de reserva com sala, data e horário, e enviar.	Reserva confirmada e exibida na interface, com mensagem de sucesso.	OK
3	Visualizar reservas do usuário	Acessar a seção “Minhas Reservas” para ver reservas ativas e canceladas.	Todas as reservas do usuário exibidas, categorizadas por status (ativas/canceladas).	OK
4	Cancelar reserva	Selecionar uma reserva ativa e cancelar.	Reserva removida da lista ativa, movida para lista de canceladas, mensagem de confirmação exibida	OK
5	Interface responsiva	Ajustar tamanho da janela para verificação de responsividade.	Interface ajusta automaticamente para diferentes tamanhos, mantendo funcionalidade e usabilidade	OK

API REST - Documentação e Integração

ID	Caso de Teste	Procedimento	Resultado Esperado	Status
1	Testar comunicação com frontend	Realizar uma reserva no frontend e verificar se a API recebe e processa a requisição corretamente.	API processa a requisição, e a reserva é criada e exibida no frontend.	OK
2	Testar resposta de erro	Realizar reserva com dados inválidos e verificar resposta da API no frontend.	API retorna erro com mensagem apropriada, exibida no frontend.	OK
3	Testar documentação da API (ex.: Postman)	Documentar endpoints no Postman e testar cada operação conforme especificações (CRUD de reservas).	Todos os endpoints documentados e testados, retornando respostas apropriadas para cada operação CRUD.	OK

5. Requisitos de Negócio - Testes de Validação

ID	Caso de Teste	Procedimento	Resultado Esperado	Status
1	Validação de horário (conflitos)	Tentar reservar uma sala já reservada para o mesmo horário.	Erro de validação com mensagem "Sala já reservada para este horário".	OK
2	Validação de reserva para o passado	Tentar reservar uma sala para um horário passado.	Erro de validação com mensagem "Reservas não podem ser feitas para o passado".	OK

6. Resultados Esperados e Conclusão

O sistema deve permitir a criação, visualização, edição e cancelamento de reservas sem conflitos de horário, além de impedir reservas para horários passados. A interface do frontend deve ser responsiva e refletir todas as operações realizadas na API.

Resultados dos testes devem ser documentados para cada caso de teste, e qualquer falha deve ser corrigida antes da entrega final.

7. Notas Finais

O relatório final deve incluir observações sobre desempenho, problemas de UX identificados, e eventuais sugestões de melhoria.