CENTRO UNIVERSITÁRIO UNIFACISA

CURSO: SISTEMAS DE INFORMAÇÃO - 2025.2

COMPETÊNCIA: ELABORAR PLANO DE TESTES E VALIDAÇÃO DE UM SOFTWARE

PROFESSOR: ANNA BEATRIZ LUCENA LIRA

NOME	MATRÍCULA
CAIO FELIPE GALDINO DA SILVA	2213080061
JOAO GABRIEL PATRIOTA ALVES DE MOURA	2213080008
LARRY DIEGO FERREIRA DE OLIVEIRA	2213080023
NICOLE LIMA CRISPIM	2213080078
THIAGO EMANUEL BARBOSA FERREIRA.	2210080021

FASE 2 – IMPLEMENTAÇÃO DA API + DOCUMENTAÇÃO DA API + IMPLEMENTAÇÃO DOS CASOS DE TESTE

Campina Grande, PB

Dezembro de 2025

1. IDENTIFICAÇÃO DO DOCUMENTO

Título: Plano de teste – Sistema de Gerenciamento de Reservas de Salas da UNIFACISA
Versão: 1.0
Versão da API: 1.0.0
Data: 03/12/2025
Responsável: Anna Lira – QA Lead, Nicole Lima – Analista de QA, Larry Diego – Analista de QA, Thiago Emanuel – Analista de QA, João Gabriel – Analista de QA, Caio Felipe – Analista de QA

1.1 LINK DA API NO GITHUB

https://github.com/LarryDiego/testes-facisa

2. RESUMO EXECUTIVO

2.1 OBJETIVO

Este relatório tem como finalidade apresentar os resultados obtidos na execução da suíte de testes automatizados do Sistema de Gerenciamento de Reservas de Salas. Foram analisados quatro módulos principais:

- Cadastro de Salas
- Cadastro de Usuários
- Gerenciador de Reservas
- Consulta de Disponibilidade

2.2 RESULTADOS GERAIS

Métrica	Quantidade	Percentual
Casos De Testes Totais	37	100%
Aprovados	35	95,12%
Reprovados	2	4,88%
Bloqueados	0	0%

2.3 COBERTURA DE CÓDIGO

A cobertura de código alcançada superou a meta para Statements e Lines, mas ficou abaixo do esperado para Branches, indicando que cenários de exceção e caminhos de decisão podem não ter sido totalmente explorados.

Métrica	Valor Alcançado	Meta (Mínima)
Statements	87,33%	>= 80%
Branches	66,29%	>= 80%
Lines	87,46%	>= 80%

2.4 STATUS POR MÓDULO

Módulo	Casos	Aprovados	Reprovados	Status

Cadastro de Salas	13	11	2	Com Pendências
Cadastro de Usuários	9	9	0	Aprovado
Gerenciamento de Reservas	12	12	0	Aprovado
Consulta de Disponibilidade	3	3	0	Aprovado

3. AMBIENTE DE TESTE

3.1 CONFIGURAÇÃO DE SOFTWARE

Tipo	Componente	Versão/Especificação	Função
Sistema Operacional	OS Host	Windows	Ambiente de execução local
Linguagem e Runtime	Node.js	v24.11.0	Ambiente de backend
Gerenciador de Pacotes	Npm	11.5.2	Gerenciamento de dependências
Framework Web	Express	4.18.2	Framework de API REST

3.2 FRAMEWORK E FERRAMENTAS DE TESTE

Tipo	Componente	Versão	Uso Principal
Framework de Testes	Jest	29.7.0	Execução dos testes e medição de cobertura
Biblioteca HTTP	Supertest	6.3.3	Simulação de requisições e respostas HTTP
Biblioteca de Validação	Express Validator	7.0.1	Validação de dados de entrada (middleware)
Persistência	In-memory	-	Simulação de banco de dados volátil

3.3 DETALHES DA EXECUÇÃO

Ferramentas Utilizadas: O Jest foi utilizado via linha de comando (npm test) no modo verbose com geração de relatório de cobertura.
Data e Hora da Execução: 22/01/2025 às 15:30:00 (UTC-3)
Duração Total: 2,274 segundos
Modo de Execução: Os dados voláteis foram reiniciados através de hooks para garantir a independência e isolamento entre os casos de teste.

4. RESULTADOS DETALHADOS

CT001 - Cadastrar sala com sucesso

Pré-condições	A API de Gerenciamento de Salas deve estar disponível. O usuário deve possuir permissão para cadastrar novas salas. Nenhuma sala com o nome “Sala 101” deve estar cadastrada previamente.

Massa de dados Parâmetros	Nome: Sala 101 Tipo: Aula Capacidade: 40 Status: ativa
Dados de teste	1. Abrir o cliente de requisições (ex: Postman ou Swagger UI). 2. Selecionar o método POST e o endpoint /salas. 3. No corpo da requisição, inserir o JSON: json { "nome": "Sala 101", "tipo": "Aula", "capacidade": 40, "status": "ativa" } 4. Enviar a requisição para o servidor. 5. Observar o código de resposta e o corpo retornado pela API.
Resultado Esperado	O sistema deve retornar o status code 201 (Created). O corpo da resposta deve conter o ID gerado automaticamente e os mesmos dados enviados. Exemplo de resposta: json { "id": 1, "nome": "Sala 101", "tipo": "Aula", "capacidade": 40, "status": "ativa" }
Resultado Obtido	
Status	APROVADO

CT002 - Impedir criação de sala com nome duplicado

Pré-condições	A API de Gerenciamento de Salas deve estar disponível. Já deve existir uma sala cadastrada com o nome “Sala 101”.
Massa de dados Parâmetros	Nome: Sala 101 Tipo: Laboratório Capacidade: 25 Status: ativa
Dados de teste	1. Abrir o cliente de requisições (ex: Postman ou Swagger UI). 2. Selecionar o método POST e o endpoint /salas. 3. No corpo da requisição, inserir o JSON: json { "nome": "Sala 101", "tipo": "Laboratório", "capacidade": 25, "status": "ativa" } 4. Enviar a requisição para o servidor. 5. Observar o código de resposta e a mensagem retornada.
Resultado Esperado	O sistema deve retornar o status code 409 (Conflict). A resposta deve conter a mensagem de erro: “Nome de sala já existente”.

Resultado Obtido	201
	Response body
	{
	"mensagem": "Sala criada com sucesso",
	"sala": {
	"id": 3,
	"nome": "sala 1",
	"tipo": "laboratório",
	"capacidade": 10,
	"status": "ativa"
	}
	}

Status	REPROVADO

CT003 - Listar todas as salas cadastradas

Pré-condições	A API de Gerenciamento de Salas deve estar disponível. Deve haver pelo menos duas salas cadastradas no sistema.
Massa de dados Parâmetros	Nenhum parâmetro necessário.
Dados de teste	1. Abrir o cliente de requisições (ex: Postman ou Swagger UI). 2. Selecionar o método GET e o endpoint /salas. 3. Enviar a requisição para o servidor. 4. Observar o código de resposta e o corpo retornado.
Resultado Esperado	O sistema deve retornar o status code 200 (OK). O corpo da resposta deve ser uma lista JSON contendo todas as salas cadastradas, com seus respectivos atributos (id, nome, tipo, capacidade, status).

Resultado Obtido	200
	Response body
	[
	{
	"id": 1,
	"nome": "sala 1",
	"tipo": "laboratorio",
	"capacidade": 10,
	"status": "ativa"
	},
	{
	"id": 2,
	"nome": "sala 1",
	"tipo": "laboratório",
	"capacidade": 10,
	"status": "ativa"
	},
	{
	"id": 3,
	"nome": "sala 1",
	"tipo": "laboratório",
	"capacidade": 10,
	"status": "ativa"
	}
	]

Status	APROVADO

CT004 - Consultar sala por ID existente

Pré-condições	A API de Gerenciamento de Salas deve estar disponível. Deve existir uma sala cadastrada com o ID 1.

Massa de dados Parâmetros	ID da sala: 1
Dados de teste	1. Abrir o cliente de requisições (ex: Postman ou Swagger UI). 2. Selecionar o método GET e o endpoint /salas/1. 3. Enviar a requisição. 4. Observar o código de resposta e o corpo retornado.
Resultado Esperado	O sistema deve retornar o status code 200 (OK). O corpo da resposta deve conter os dados da sala com o ID informado, por exemplo: json { "id": 1, "nome": "Sala 101", "tipo": "Aula", "capacidade": 40, "status": "ativa" }
Resultado Obtido	
Status	APROVADO

CT005 - Consultar sala por ID inexistente

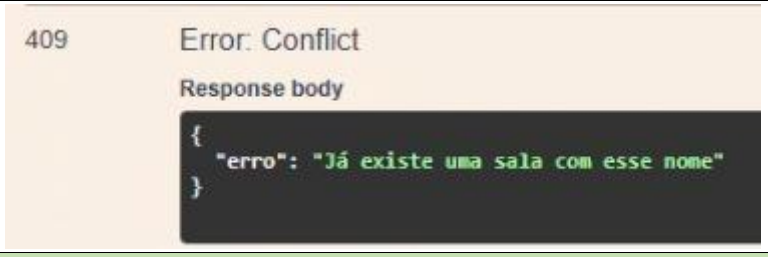
Pré-condições	A API de Gerenciamento de Salas deve estar disponível. Nenhuma sala deve estar cadastrada com o ID 999.
Massa de dados Parâmetros	ID da sala: 999
Dados de teste	1. Abrir o cliente de requisições (ex: Postman ou Swagger UI). 2. Selecionar o método GET e o endpoint /salas/999. 3. Enviar a requisição. 4. Observar o código de resposta e o corpo retornado.
Resultado Esperado	O sistema deve retornar o status code 404 (Not Found). A resposta deve conter a mensagem “Sala não encontrada”.
Resultado Obtido	
Status	APROVADO

CT006 - Atualizar informações de uma sala existente

Pré-condições	A API de Gerenciamento de Salas deve estar disponível. Deve existir uma sala cadastrada com o ID 1 (ex: “Sala 101”).
Massa de dados Parâmetros	ID da sala: 1 Tipo: Laboratório Capacidade: 35

Dados de teste	1. Abrir o cliente de requisições (ex: Postman ou Swagger UI). 2. Selecionar o método PUT e o endpoint /salas/1. 3. No corpo da requisição, inserir o JSON: json { "tipo": "Laboratório", "capacidade": 35 } 4. Enviar a requisição para o servidor. 5. Observar o código de resposta e o corpo retornado.
Resultado Esperado	O sistema deve retornar o status code 200 (OK). O corpo da resposta deve conter os dados atualizados da sala: json { "id": 1, "nome": "Sala 101", "tipo": "Laboratório", "capacidade": 35, "status": "ativa" }
Resultado Obtido	
Status	APROVADO

CT007 - Impedir atualização de sala com nome duplicado

Pré-condições	A API de Gerenciamento de Salas deve estar disponível. Devem existir duas salas cadastradas: “Sala 101” (ID 1) e “Sala 102” (ID 2).
Massa de dados Parâmetros	ID da sala: 2 Novo nome: Sala 101
Dados de teste	1. Abrir o cliente de requisições (ex: Postman ou Swagger UI). 2. Selecionar o método PUT e o endpoint /salas/2. 3. No corpo da requisição, inserir o JSON: json { "nome": "Sala 101" } 4. Enviar a requisição. 5. Observar o código de resposta e a mensagem de retorno.
Resultado Esperado	O sistema deve retornar o status code 409 (Conflict). O corpo da resposta deve conter a mensagem: “Nome de sala já existente”.
Resultado Obtido	
Status	APROVADO

CT008 - Remover sala existente

Pré-condições	A API de Gerenciamento de Salas deve estar disponível. Deve existir uma sala cadastrada com o ID 3.
Massa de dados Parâmetros	ID da sala: 3
Dados de teste	1. Abrir o cliente de requisições (ex: Postman ou Swagger UI). 2. Selecionar o método DELETE e o endpoint /salas/3. 3. Enviar a requisição. 4. Observar o código de resposta. 5. Em seguida, realizar um GET /salas/3 para verificar se a sala foi realmente removida.
Resultado Esperado	O sistema deve retornar o status code 204 (No Content). Ao tentar consultar novamente a sala, o sistema deve retornar 404 (Not Found).
Resultado Obtido	
Status	APROVADO

CT009 - Remover sala inexistente

Pré-condições	A API de Gerenciamento de Salas deve estar disponível. Nenhuma sala cadastrada deve possuir o ID 999.
Massa de dados Parâmetros	ID da sala: 999
Dados de teste	1. Abrir o cliente de requisições (ex: Postman ou Swagger UI). 2. Selecionar o método DELETE e o endpoint /salas/999. 3. Enviar a requisição. 4. Observar o código de resposta e a mensagem de erro retornada.
Resultado Esperado	O sistema deve retornar o status code 404 (Not Found). O corpo da resposta deve conter a mensagem: “Sala não encontrada”.
Resultado Obtido	
Status	APROVADO

CT010 - Criar sala com status inativa

Pré-condições	A API de Gerenciamento de Salas deve estar disponível. O nome “Sala Reunião 01” não deve estar cadastrado.
Massa de dados Parâmetros	Nome: Sala Reunião 01 Tipo: Reunião Capacidade: 10 Status: inativa
Dados de teste	1. Abrir o cliente de requisições (ex: Postman ou Swagger UI). 2. Selecionar o método POST e o endpoint /salas. 3. No corpo da requisição, inserir o JSON: json { "nome": "Sala Reunião 01", "tipo": "Reunião", "capacidade": 10, "status": "inativa" } 4. Enviar a requisição. 5. Observar o código de resposta e o corpo retornado.
Resultado Esperado	O sistema deve retornar o status code 201 (Created). O corpo da resposta deve conter os dados da sala criada, com o campo "status": "inativa".
Resultado Obtido	
Status	APROVADO

CT011 - Verificar que sala inativa não pode ser reservada

Pré-condições	A API de Gerenciamento de Salas e a API de Reservas devem estar disponíveis. Deve existir uma sala cadastrada com o nome “Sala Reunião 01” e status “inativa”.
Massa de dados Parâmetros	Endpoint: POST /reservas salaId: 5 usuário: prof.joao data: 2025-11-15T10:00:00
Dados de teste	1. Abrir o cliente de requisições (ex: Postman ou Swagger UI). 2. Selecionar o método POST e o endpoint /reservas. 3. Inserir no corpo da requisição o JSON: json { "salaId": 5, "usuario": "prof.joao", "data": "2025-11-15T10:00:00" } 4. Enviar a requisição. 5. Observar o código de resposta e a mensagem retornada.
Resultado Esperado	O sistema deve retornar o status code 400 (Bad Request).

	O corpo da resposta deve conter a mensagem: “Sala inativa não pode ser reservada”.
Resultado Obtido	
Status	APROVADA

CT012 - Criar sala com campo obrigatório ausente

Pré-condições	A API de Gerenciamento de Salas deve estar disponível.
Massa de dados Parâmetros	Nome: (ausente) Tipo: Laboratório Capacidade: 25 Status: ativa
Dados de teste	1. Abrir o cliente de requisições (ex: Postman ou Swagger UI). 2. Selecionar o método POST e o endpoint /salas. 3. Inserir no corpo da requisição o JSON sem o campo nome: json { "tipo": "Laboratório", "capacidade": 25, "status": "ativa" } 4. Enviar a requisição. 5. Observar o código de resposta e a mensagem retornada.
Resultado Esperado	O sistema deve retornar o status code 400 (Bad Request). O corpo da resposta deve conter uma mensagem informando que o campo “nome” é obrigatório.
Resultado Obtido	
Status	APROVADO

CT013 - Criar sala com capacidade inválida

Pré-condições	A API de Gerenciamento de Salas deve estar disponível.
Massa de dados Parâmetros	Nome: Sala Teste Tipo: Aula Capacidade: 0 Status: ativa

Dados de teste	1. Abrir o cliente de requisições (ex: Postman ou Swagger UI). 2. Selecionar o método POST e o endpoint /salas. 3. Inserir o corpo da requisição com capacidade igual a 0: json { "nome": "Sala Teste", "tipo": "Aula", "capacidade": 0, "status": "ativa" } 4. Enviar a requisição. 5. Observar o código de resposta e a mensagem retornada.
Resultado Esperado	O sistema deve retornar o status code 400 (Bad Request). O corpo da resposta deve conter a mensagem: “Capacidade deve ser maior que zero”.
Resultado Obtido	
Status	REPROVADO

4.1 CADASTRO DE USUÁRIO

CT014 - Cadastrar usuário com sucesso

Pré-condições	A API de Usuários deve estar disponível. O e-mail informado não deve estar cadastrado.
Massa de dados Parâmetros	Nome: João Silva E-mail: joao.silva@unifacisa.edu.br
Dados de teste	1. Abrir o cliente de requisições (Postman ou Swagger UI). 2. Selecionar o método POST e o endpoint /usuarios. 3. Inserir o corpo da requisição:
	json { "nome": "João Silva", "email": "joao.silva@unifacisa.edu.br" } 4. Enviar a requisição. 5. Observar o código de resposta e o corpo retornado.json { "nome": "Sala Teste", "tipo": "Aula", "capacidade": 0, "status": "ativa" } 4. Enviar a requisição. 5. Observar o código de resposta e a mensagem retornada.
Resultado Esperado	O sistema deve retornar o status code 400 (Bad Request). O corpo da resposta deve conter a mensagem: “Capacidade deve ser maior que zero”.

Resultado Obtido	
Status	APROVADO

CT015 - Impedir cadastro com e-mail duplicado

Pré-condições	A API deve estar disponível. Já existir um usuário cadastrado com o e-mail “joao.silva@unifacisa.edu.br”.
Massa de dados Parâmetros	Nome: João Silva E-mail: joao.silva@unifacisa.edu.br
Dados de teste	1. Selecionar o método POST e o endpoint /usuarios. 2. Inserir o corpo da requisição: json { "nome": "João Silva", "email": "joao.silva@unifacisa.edu.br" } 3. Enviar a requisição. 4. Observar a resposta e o código retornado.
Resultado Esperado	O sistema deve retornar 409 (Conflict). O corpo deve conter mensagem: “E-mail já cadastrado.”
Resultado Obtido	
Status	APROVADO

CT016 - Impedir cadastro com e-mail inválido

Pré-condições	A API deve estar disponível. Já existir um usuário cadastrado com o e-mail “joao.silva@unifacisa.edu.br”.
Massa de dados Parâmetros	Nome: Maria Souza E-mail: maria.souza@unifacisa (inválido)
Dados de teste	1. Selecionar o método POST e o endpoint /usuarios. 2. Inserir o corpo da requisição: json { "nome": "Maria Souza", "email": "maria.souza@unifacisa" } 3. Enviar a requisição. 4. Observar o código e mensagem retornada.

Resultado Esperado	O sistema deve retornar 400 (Bad Request). O corpo deve conter mensagem: “E-mail inválido.”
Resultado Obtido	
Status	APROVADO

CT017 - Listar todos os usuários cadastrados

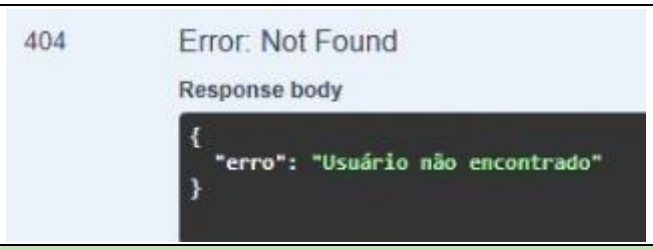
Pré-condições	A API deve estar disponível.
	Devem existir usuários cadastrados no sistema.
Massa de dados Parâmetros	Nenhum
Dados de teste	1. Selecionar o método GET e o endpoint /usuarios. 2. Enviar a requisição. 3. Observar o código e o corpo da resposta.
Resultado Esperado	O sistema deve retornar 200 (OK). O corpo deve conter uma lista JSON com todos os usuários cadastrados.
Resultado Obtido	
Status	APROVADO

CT018 - Consultar usuário por ID existente

Pré-condições	A API deve estar disponível. Deve existir um usuário com ID 1 cadastrado.
Massa de dados Parâmetros	ID do usuário: 1

Dados de teste	1. Selecionar o método GET e o endpoint /usuarios/1. 2. Enviar a requisição. 3. Observar o código de resposta e o corpo retornado.
Resultado Esperado	O sistema deve retornar 200 (OK). O corpo deve conter os dados do usuário com ID 1.
Resultado Obtido	
Status	APROVADO

CT019 - Consultar usuário por ID inexistente

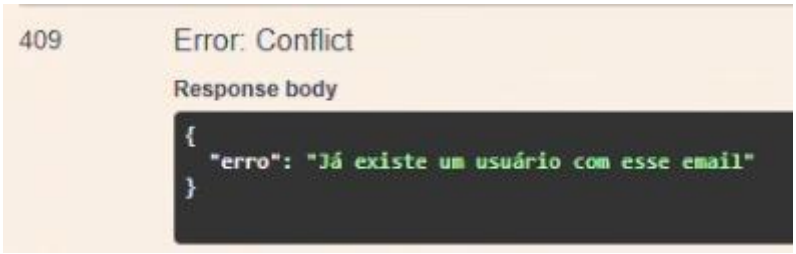
Pré-condições	A API deve estar disponível. Nenhum usuário deve estar cadastrado com o ID 999.
Massa de dados Parâmetros	ID do usuário: 999
Dados de teste	1. Selecionar o método GET e o endpoint /usuarios/999. 2. Enviar a requisição. 3. Observar o código de resposta e o corpo retornado.
Resultado Esperado	O sistema deve retornar 404 (Not Found). A resposta deve conter a mensagem: “Usuário não encontrado.”
Resultado Obtido	
Status	APROVADO

CT020 - Atualizar informações de um usuário existente

Pré-condições	A API deve estar disponível. Deve existir um usuário cadastrado com ID 1.
Massa de dados Parâmetros	ID do usuário: 1 Novo nome: João Pedro Novo e-mail: joao.pedro@unifacisa.edu.br
Dados de teste	1. Selecionar o método PUT e o endpoint /usuarios/1. 2. Inserir o corpo da requisição: json { "nome": "João Pedro", "email": "joao.pedro@unifacisa.edu.br" } 3. Enviar a requisição.
	4. Observar o código e o corpo retornado.

Resultado Esperado	O sistema deve retornar 200 (OK). O corpo deve exibir os dados atualizados do usuário.
Resultado Obtido	
Status	APROVADO

CT021 - Impedir atualização com e-mail duplicado

Pré-condições	A API deve estar disponível. Devem existir dois usuários: João (ID 1, e-mail joao@unifacisa.edu.br) e Maria (ID 2, e-mail maria@unifacisa.edu.br).
Massa de dados Parâmetros	ID do usuário: 2 Novo e-mail: joao@unifacisa.edu.br
Dados de teste	1. Selecionar o método PUT e o endpoint /usuarios/2. 2. Inserir o corpo da requisição: json { "email": "joao@unifacisa.edu.br" } 3. Enviar a requisição. 4. Observar o código de resposta e a mensagem retornada.
Resultado Esperado	O sistema deve retornar 409 (Conflict). O corpo deve conter a mensagem: “E-mail já cadastrado.”
Resultado Obtido	
Status	APROVADO

CT022 - Remover usuário existente

Pré-condições	A API deve estar disponível Deve existir um usuário cadastrado com ID 3.
Massa de dados Parâmetros	ID do usuário: 3
Dados de teste	1. Selecionar o método DELETE e o endpoint /usuarios/3. 2. Enviar a requisição. 3. Observar o código de resposta. 4. Fazer um GET /usuarios/3 para confirmar a exclusão.
Resultado Esperado	O sistema deve retornar 204 (No Content). Ao consultar novamente, deve retornar 404 (Not Found).

Resultado Obtido	200
	Response body
	<pre>{
	  "mensagem": "Usuário removido com sucesso",
	  "usuario": {
	  "id": 2,
	  "nome": "Larry",
	  "email": "larry@example.com"
	  }
	}</pre>

Status	APROVADO

4.2 GERENCIAMENTO DE RESERVAS

CT023 - Criar reserva com sucesso

Pré-condições	A API deve estar disponível. O usuário e a sala informados devem existir. Não deve haver conflito de horários para a sala.
Massa de dados Parâmetros	usuario_id: 1 sala_id: 2 data: 2025-11-10 hora_inicio: 09:00 hora_fim: 10:00 motivo: Reunião de planejamento
Dados de teste	1. Selecionar o método POST e o endpoint /reservas. 2. Inserir o corpo da requisição:
	json { "usuario_id": 1, "sala_id": 2, "data": "2025-11-10", "hora_inicio": "09:00", "hora_fim": "10:00", "motivo": "Reunião de planejamento" } 3. Enviar a requisição. 4. Observar o código e o corpo da resposta.
Resultado Esperado	O sistema deve retornar 201 (Created). O corpo deve conter o ID gerado e os dados da reserva criada.

Resultado Obtido	201
	Response body
	<pre>{
	  "mensagem": "Reserva criada com sucesso",
	  "reserva": {
	  "id": 2,
	  "usuario_id": 1,
	  "sala_id": 2,
	  "data": "2025-12-15",
	  "hora_inicio": "14:00",
	  "hora_fim": "16:00",
	  "motivo": "string"
	  }
	}</pre>

Status	APROVADO

CT024 - Impedir criação de reserva com horários sobrepostos

Précondições	Deve existir uma reserva cadastrada na sala 2 das 09:00 às 10:00 no dia 2025-11-10.

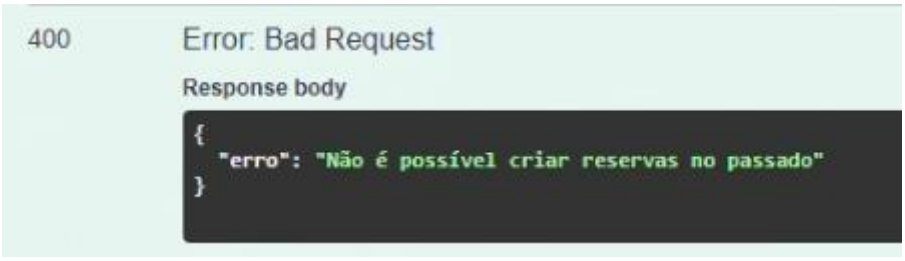
Massa de dados Parâmetros	usuario_id: 3 sala_id: 2 data: 2025-11-10 hora_inicio: 09:30 hora_fim: 10:30 motivo: Treinamento
Dados de teste	1. Selecionar o método POST e o endpoint /reservas. 2. Inserir o corpo: json { "usuario_id": 3, "sala_id": 2, "data": "202511-10", "hora_inicio": "09:30", "hora_fim": "10:30", "motivo": "Treinamento" } 3. Enviar a requisição.
Resultado Esperado	O sistema deve retornar 409 (Conflict). Mensagem: “Horário indisponível para esta sala.”
Resultado Obtido	
Status	APROVADO

CT025 - Impedir criação de reserva com horários sobrepostos

Pré-condições	A API deve estar disponível.
Massa de dados Parâmetros	usuario_id: 2 sala_id: 1 data: 2025-11-11 hora_inicio: 15:00 hora_fim: 14:00 motivo: Teste inválido
Dados de teste	1. Selecionar o método POST e o endpoint /reservas. 2. Inserir o corpo: json { "usuario_id": 2, "sala_id": 1, "data": "2025-11-11", "hora_inicio": "15:00", "hora_fim": "14:00", "motivo": "Teste inválido" } 3. Enviar a requisição.
Resultado Esperado	O sistema deve retornar 400 (Bad Request). Mensagem: “A hora de término deve ser maior que a hora de início.”
Resultado Obtido	
Status	

CT026 - Impedir criação de reserva em data passada

Précondições	A API deve estar disponível.

Massa de dados Parâmetros	usuario_id: 1 sala_id: 2 data: 2025-01-10 (data passada) hora_inicio: 09:00 hora_fim: 10:00 motivo: Teste de data antiga
Dados de teste	1. Selecionar o método POST e o endpoint /reservas. 2. Inserir o corpo: json { "usuario_id": 1, "sala_id": 2, "data": "2025-01-10", "hora_inicio": "09:00", "hora_fim": "10:00", "motivo": "Teste de data antiga" } 3. Enviar a requisição.
Resultado Esperado	O sistema deve retornar 400 (Bad Request). Mensagem: “Não é permitido criar reservas em datas passadas.”
Resultado Obtido	
Status	APROVADO

CT027 - Listar todas as reservas cadastradas

Pré-condições	A API deve estar disponível. Devem existir reservas cadastradas.
Massa de dados Parâmetros	Nenhum
Dados de teste	1. Selecionar o método GET e o endpoint /reservas. 2. Enviar a requisição. 3. Observar o código e o corpo da resposta.
Resultado Esperado	O sistema deve retornar 200 (OK). O corpo deve conter uma lista JSON com as reservas existentes.

Resultado Obtido	200
	Response body
	[
	{
	"id": 1,
	"usuario_id": 1,
	"sala_id": 3,
	"data": "2025-12-15",
	"hora_inicio": "14:00",
	"hora_fim": "16:00",
	"motivo": "string"
	},
	{
	"id": 2,
	"usuario_id": 1,
	"sala_id": 2,
	"data": "2025-12-15",
	"hora_inicio": "14:00",
	"hora_fim": "16:00",
	"motivo": "string"
	}
	]

Status	APROVADO

CT028 - Consultar reserva por ID existente

Pré-condições	Deve existir uma reserva cadastrada com ID 1.
Massa de dados Parâmetros	ID da reserva: 1
Dados de teste	1. Selecionar o método GET e o endpoint /reservas/1. 2. Enviar a requisição. 3. Observar o retorno.
Resultado Esperado	O sistema deve retornar 200 (OK). O corpo deve conter os detalhes da reserva com ID 1.

Resultado Obtido	200
	Response body
	{
	"id": 2,
	"usuario_id": 1,
	"sala_id": 2,
	"data": "2025-12-15",
	"hora_inicio": "14:00",
	"hora_fim": "16:00",
	"motivo": "string"
	}

Status	APROVADO

CT029 - Consultar reservas de uma sala em uma data específica

Pré-condições	Deve existir ao menos uma reserva na sala 2 para o dia 202511-10.
Massa de dados Parâmetros	sala_id: 2 data: 2025-11-10
Dados de teste	1. Selecionar o método GET e o endpoint /reservas?sala_id=2&data=2025-11-10.
	2. Enviar a requisição. 3. Observar o retorno.

Resultado Esperado	O sistema deve retornar 200 (OK). O corpo deve listar todas as reservas da sala 2 nessa data.
Resultado Obtido	
Status	APROVADO

CT030 - Consultar reservas de um usuário específico

Pré-condições	Deve existir ao menos uma reserva criada pelo usuário 1.
Massa de dados Parâmetros	usuario_id: 1
Dados de teste	1. Selecionar o método GET e o endpoint /reservas?usuario_id=1. 2. Enviar a requisição. 3. Observar o retorno.
Resultado Esperado	O sistema deve retornar 200 (OK). O corpo deve listar todas as reservas criadas pelo usuário informado.
Resultado Obtido	
Status	APROVADO

CT031 - Atualizar informações de uma reserva existente

Pré-condições	Deve existir uma reserva com ID 5 cadastrada.
Massa de dados Parâmetros	ID da reserva: 5 hora_inicio: 14:00 hora_fim: 15:30 motivo: Ajuste de horário

Dados de teste	1. Selecionar o método PUT e o endpoint /reservas/5. 2. Inserir o corpo: json { "hora_inicio": "14:00", "hora_fim": "15:30", "motivo": "Ajuste de horário" } 3. Enviar a requisição.
Resultado Esperado	O sistema deve retornar 200 (OK). O corpo deve conter os dados atualizados da reserva.
Resultado Obtido	
Status	APROVADO

CT032 - Impedir atualização que gere sobreposição de horário

Précondições	Deve existir outra reserva na mesma sala no intervalo 14:00– 15:00. Deve existir uma reserva a ser atualizada que entraria nesse horário.
Massa de dados Parâmetros	ID da reserva: 6
Dados de teste	1. Selecionar o método PUT e o endpoint /reservas/6. 2. Inserir o corpo: json { "hora_inicio": "14:00", "hora_fim": "15:00" } 3. Enviar a requisição.
Resultado Esperado	O sistema deve retornar 409 (Conflict). Mensagem: “Horário indisponível para esta sala.”
Resultado Obtido	
Status	APROVADO

CT033 - Cancelar reserva antes do horário de início

Pré-condições	Deve existir uma reserva com início futuro (ex.: amanhã às 10:00).
Massa de dados Parâmetros	ID da reserva: 7

Dados de teste	1. Selecionar o método DELETE e o endpoint /reservas/7. 2. Enviar a requisição. 3. Consultar novamente a reserva.
Resultado Esperado	O sistema deve retornar 204 (No Content). A reserva deve ser removida com sucesso.

Resultado Obtido	200
	Response body
	{
	"mensagem": "Reserva cancelada com sucesso",
	"reserva": {
	"id": 2,
	"usuario_id": 1,
	"sala_id": 2,
	"data": "2025-12-30",
	"hora_inicio": "14:00",
	"hora_fim": "15:00",
	"motivo": "string"
	}
	}

Status	APROVADO

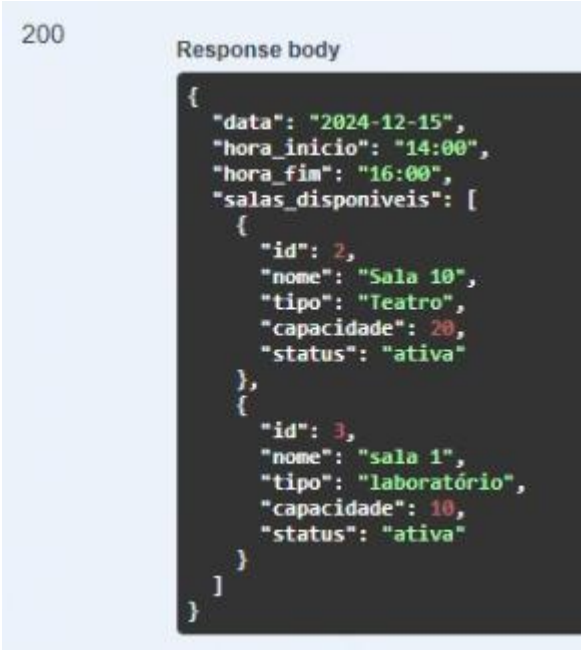
CT034 - Impedir cancelamento de reserva após horário de início

Pré-condições	Deve existir uma reserva cujo horário de início já tenha passado.
Massa de dados Parâmetros	ID da reserva: 8
Dados de teste	1. Selecionar o método DELETE e o endpoint /reservas/8. 2. Enviar a requisição. 3. Observar o retorno.
Resultado Esperado	O sistema deve retornar 400 (Bad Request). Mensagem: “Não é permitido cancelar reservas após o horário de início.”
Resultado Obtido	{ "mensagem": "Não é possível cancelar após o horário de início" }
Status	APROVADO

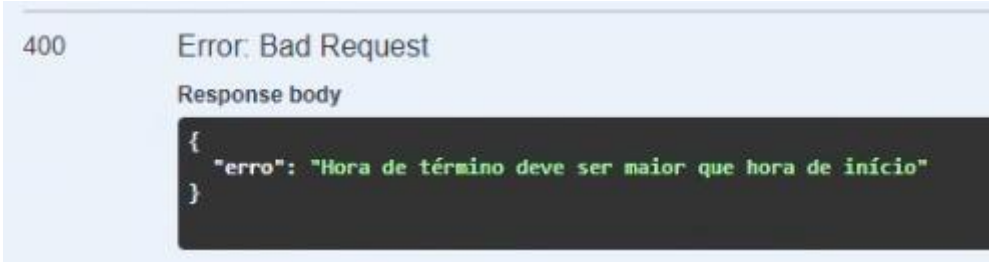
4.3 CONSULTA DE DISPONIBILIDADE

CT035 - Consultar salas disponíveis com sucesso

Pré-condições	A API deve estar disponível. Devem existir salas cadastradas, algumas reservadas e outras livres para o período.
Massa de dados Parâmetros	data: 2025-11-12 hora_inicio: 09:00 hora_fim: 10:00
Dados de teste	1. Selecionar o método GET e o endpoint /salas/disponiveis?data=2025-1112&hora_inicio=09:00&hora_fim=10:00. 2. Enviar a requisição. 3. Observar o código e o corpo da resposta.

Resultado Esperado	O sistema deve retornar 200 (OK). O corpo da resposta deve listar todas as salas que não possuem reservas no período informado. Exemplo de retorno: json [{ "id": 1, "nome": "Sala 101", "status": "ativa" }, { "id": 3, "nome": "Auditório", "status": "ativa" }]
Resultado Obtido	
Status	APROVADO

CT036 - Impedir consulta com hora_fim menor que hora_inicio

Précondições	A API deve estar disponível.
Massa de dados Parâmetros	data: 2025-11-14 hora_inicio: 15:00 hora_fim: 14:00
Dados de teste	1. Selecionar o método GET e o endpoint /salas/disponiveis?data=2025-1114&hora_inicio=15:00&hora_fim=14:00. 2. Enviar a requisição. 3. Observar o retorno.
Resultado Esperado	O sistema deve retornar 400 (Bad Request). Mensagem: “A hora de término deve ser maior que a hora de início.”
Resultado Obtido	
Status	APROVADO

CT037 - Impedir consulta sem parâmetros obrigatório

Précondições	A API deve estar disponível.
Massa de dados Parâmetros	data: (ausente) hora_inicio: 08:00
	hora_fim: 09:00
Dados de teste	1. Selecionar o método GET e o endpoint /salas/disponiveis?hora_inicio=08:00&hora_fim=09:00 (sem parâmetro data). 2. Enviar a requisição. 3. Observar o retorno.
Resultado Esperado	O sistema deve retornar 400 (Bad Request). ● Mensagem: “Parâmetro obrigatório ausente: data.”
Resultado Obtido	
Status	APROVADO

5. RESULTADOS DETALHADOS

Os defeitos encontrados são de natureza funcional e de validação de dados, sendo classificados como CRÍTICOS (P1 - Prioridade Alta) por comprometerem a integridade do sistema.

5.1 DEFEITO 1: DEF-001 - VIOLAÇÃO DE UNICIDADE DE NOME DE SALA

- **ID e Severidade:** DEF - 001 - CRÍTICA
- **Caso de Teste Relacionado:** CT002 - Impedir criação de sala com nome duplicado
- **Descrição do Problema:** O sistema falha em aplicar a Regra de Negócio RN-001 (Nome de Sala Único). O endpoint POST /api/salas retorna 201 (Created) e persiste a sala duplicada, violando a integridade do cadastro.
- Passos para Reproduzir:
- POST /api/salas (Criar "Sala 101"): Resultado 201.
- POST /api/salas (Tentar recriar "Sala 101"): Resultado 201.
- **Causa Raiz:** A validação de nome único está comentada no método criar() do Model Sala.js.
- **Status:** ABERTO
- **Impacto:** Integridade de Dados Comprometida; Confusão Operacional.

5.2 DEFEITO 2: DEF-002 - ACEITAÇÃO DE CAPACIDADE ZERO EM SALA

- ID e Severidade: DEF - 002 - CRÍTICA
- **Caso de Teste Relacionado:** CT013 - Criar sala com capacidade inválida
- **Descrição do Problema:** O sistema permite capacidade: 0 ao criar uma sala, o que é logicamente inconsistente com o propósito de uma sala reservável. O sistema retorna 201 (Created), quando o esperado era 400 (Bad Request) e uma mensagem de erro de validação.
- **Passos para Reproduzir:**
  1. POST /api/salas com {"capacidade": 0}: Resultado 201.
- Causa Raiz: O middleware de validação (salaValidations.js) foi configurado com isInt({ min: 0 }), permitindo o valor zero. O correto seria min: 1.
- Status: ABERTO
- Impacto: Violação da Lógica de Negócio; Inconsistência em Relatórios de Capacidade.

6. CONCLUSÃO

6.1 AVALIAÇÃO GERAL DA QUALIDADE

Apesar da alta taxa de sucesso (95,12%) e da excelente cobertura funcional nos módulos de Usuários e Reservas, os defeitos DEF-001 e DEF-002 no módulo de Cadastro de Salas são bloqueadores de deploy. Estes problemas demonstram falhas nas validações de regras de negócio fundamentais. O sistema não está liberado para o ambiente de produção até a correção desses pontos.

6.2 PROBLEMAS ENCONTRADOS

- **Defeitos Críticos:** 2 abertos (DEF-001 e DEF-002).
- **Baixa Cobertura de Branches:** Atingiu 66,29%, abaixo da meta de 80%, sugerindo que alguns fluxos de erro não estão sendo testados.

6.3 RECOMENDAÇÕES E PRÓXIMOS PASSOS

PRIORIDADE	AÇÃO	DETALHES TÉCNICOS
P1 - URGENTE	Correção Imediata dos Defeitos Críticos	Reativar a validação de nome único (Sala.js) e ajustar a validação de capacidade (salaValidations.js para min: 1).
P1 – URGENTE	Re-execução de Testes (Regressão)	Executar a suíte de 37 CTs novamente. O critério de aceite é 100% de sucesso.
P2 - ALTA	Melhoria da Cobertura de Branches	Adicionar testes unitários para cobrir as linhas de código não executadas nos Controllers e Models (linhas não cobertas em ANEXO B)
P2 - ALTA	Implementação de Validação Dupla	Garantir que as validações existam tanto no middleware (para erro rápido

		HTTP 400) quanto no model (para proteção interna e integridade).
P3 - MÉDIA	Evolução da Persistência	Planejar a migração da persistência inmemory para um SGBD real (e.g., PostgreSQL) com constraints de unicidade e não-nulo implementadas a nível de banco de dados.