INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO SERTÃO PERNAMBUCANO – CAMPUS SALGUEIRO

VISÃO DO PRODUTO

Versão 0.1

28/07/2025

Caren Beatriz Silva Oliveira

Daiane Maria dos Santos Ribeiro

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO SERTÃO PERNAMBUCANO – CAMPUS SALGUEIRO

HISTÓRICO

Data	Versão	Descrição	Autor
28/07/2025	0.1	Documento de visão inicial	Caren Daiane

SUMÁRIO

1. INTRODUÇÃO

O presente documento tem como objetivo principal apresentar a visão do sistema que será desenvolvido na disciplina de Programação Web II, do curso superior de Tecnologia em Sistemas para Internet do IFSertãoPE campus Salgueiro. O sistema em questão consiste em uma *Application Programming Interface* (API) para reservas de salas, implementada com Java EE, utilizando os frameworks *Spring Boot, Spring Data JPA* e *Hibernate*, além de integrar o banco de dados *MySQL Workbench*. A referida API será responsável por fornecer as funcionalidades necessárias para o agendamento, consulta e gerenciamento de reservas de salas, garantindo organização, disponibilidade e facilidade de uso para os usuários.

Sendo assim, esse documento de visão apresenta detalhes sobre a modelagem do sistema, uso da arquitetura REST, padrões do projeto, e as tecnologias utilizadas.

1.1 Escopo

O sistema permitirá que os usuários cadastrem salas, reservem horários disponíveis, cancelem reservas e consultem a disponibilidade dos espaços. O público-alvo são servidores e administradores da instituição de ensino que necessitam reservar espaços físicos para reuniões, aulas, palestras, eventos entre outros.

As funcionalidades principais do sistema são:

- Cadastro de usuários e autenticação.
- Cadastro e listagem de salas.
- Realização, alteração e cancelamento de reservas.
- Consulta da disponibilidade de horários.
- Controle de permissões de acesso (administrador e usuário comum).

1.2 Definições, Acrônimos e Abreviações

API : Application Programming Interface
CRUD : Create, Read, Update, Delete
DTO: Data Transfer Object
Java EE : Java Platform, Enterprise Edition
JPA: Java Persistence API
JSON:JavaScript Object Notation
JWT: JSON Web Token
ORM: Object-Relational Mapping
REST: Representational State Transfer
RF : Requisito Funcional
RNF : Requisito Não Funcional

SQL: Structured Query Language
UC : User Case (Caso de Uso)
URLs: Uniform Resource Locator

2. MODELAGEM DE SISTEMAS

A modelagem do sistema API REST para reserva de salas foi realizada com base na identificação das principais funcionalidades que devem estar presentes no referido sistema, utilizando os conceitos de orientação a objetos e arquitetura REST. Abaixo, são apresentados os principais casos de uso e a estrutura do modelo de objetos, que servem de base para o desenvolvimento da API.

2.1 Casos de Uso

Caso de Uso: Cadastro de Usuário

Descrição: Permite que um novo usuário se registre no sistema com nome, email e senha.

Cenário Principal:

1. O usuário acessa a tela de cadastro.
2. Preenche os dados obrigatórios.
3. O sistema valida os dados e salva no banco de dados.

Fluxo Alternativo:

- Dados inválidos ou email já registrado.

Exceções:

- Falha de conexão com o banco de dados.

Caso de Uso: Autenticação de Usuário

Descrição: Permite que o usuário realize login com email e senha cadastrados.

Cenário Principal:

1. O usuário informa email e senha.
2. O sistema valida as credenciais.
3. Se válidas, é gerado um token de autenticação.

Fluxo Alternativo:

- Senha ou email incorretos.

Exceções:

- Erro no servidor de autenticação.

Caso de Uso: Cadastro de Sala (Administrador)

Descrição: Permite ao administrador cadastrar uma nova sala no sistema.

Cenário Principal:

1. O administrador acessa a tela de cadastro de sala.
2. Informa nome, local e capacidade.
3. O sistema salva a sala no banco.

Fluxo Alternativo:

- Dados incompletos ou inválidos.

Exceções:

- Tentativa de cadastrar sala duplicada.

Caso de Uso: Realizar Reserva de Sala

Descrição: Permite que o usuário selecione uma sala e reserve um horário.

Cenário Principal:

1. O usuário autentica-se no sistema.
2. O sistema apresenta as salas disponíveis.
3. O usuário escolhe uma sala e um horário livre.
4. O sistema confirma a reserva.

Fluxo Alternativo:

- Caso o horário esteja indisponível, o sistema informa o conflito e solicita nova seleção.

Exceções:

- Tentativa de reserva em horário já ocupado.

Caso de Uso: Alterar Reserva

Descrição: Permite que o usuário edite uma reserva já realizada.

Cenário Principal:

1. O usuário acessa suas reservas.
2. Seleciona a reserva a ser modificada.
3. Escolhe novos horários disponíveis.
4. O sistema salva a alteração.

Fluxo Alternativo:

- Novos horários indisponíveis.

Exceções:

- Tentativa de editar reserva de outro usuário.

Caso de Uso: Cancelar Reserva

Descrição: Permite que o usuário cancele uma reserva.

Cenário Principal:

1. O usuário acessa suas reservas.
2. Seleciona a reserva desejada.
3. Solicita o cancelamento.

Fluxo Alternativo:

- Cancelamento de reserva já expirada.

Exceções:

- Falha na operação de remoção do banco.

Caso de Uso: Consultar Disponibilidade

Descrição: Permite que o usuário veja os horários disponíveis para uma sala.

Cenário Principal:

1. O usuário acessa a tela de busca de disponibilidade.
2. Seleciona uma sala e uma data.
3. O sistema exibe os horários disponíveis.

Fluxo Alternativo:

- Nenhuma sala disponível no período.

Exceções:

- Falha na comunicação com o banco de dados.

2.2 Modelo de Objetos

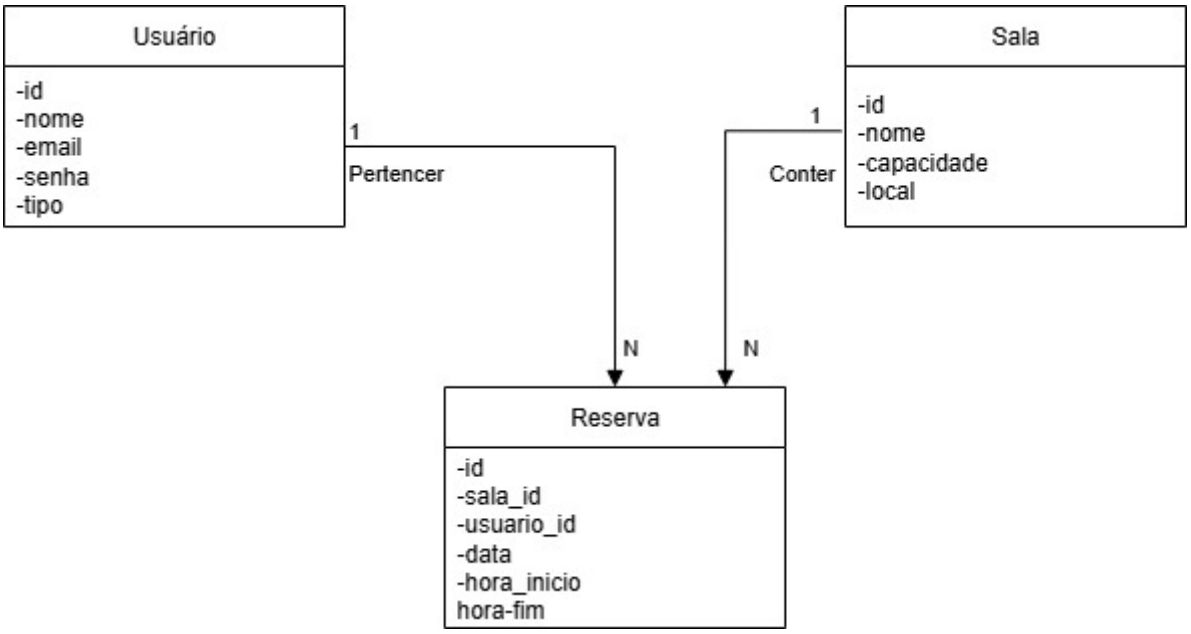
O sistema possui as seguintes entidades principais:

- **Usuário:** Atributos: id, nome, email, senha, tipo (admin/comum)
- **Sala:** Atributos: id, nome, capacidade, local
- **Reserva:** Atributos: id, sala_id, usuario_id, data, hora_inicio, hora_fim

O relacionamento entre as entidades é o seguinte:

- Um usuário pode fazer muitas reservas.
- Uma sala pode conter muitas reservas.
- Cada reserva pertence a um único usuário e a uma única sala.

A seguir apresentaremos o Diagrama de Classes UML contendo os atributos e relacionamentos, seguindo o padrão da orientação a objetos.



3. MODELO DE ARQUITETURA REST

A arquitetura REST é uma forma simples de organizar os caminhos (URLs) da nossa API. Cada funcionalidade do sistema terá um endereço específico e uma ação (como listar, cadastrar, alterar ou excluir - CRUD). Por exemplo:

- POST /usuarios — Cadastrar novo usuário
- POST /auth/login — Fazer login
- GET /salas — Listar salas
- POST /salas — Cadastrar sala (somente administrador)
- GET /reservas — Ver reservas do usuário
- POST /reservas — Criar uma reserva
- PUT /reservas/{id} — Editar uma reserva
- DELETE /reservas/{id} — Cancelar uma reserva
- GET /disponibilidade?sala_id=1&data=2025-01-01 — Verificar horários disponíveis

Os dados serão enviados e recebidos no formato JSON. Para proteger a API, será usado login com token (JWT), que permite o acesso às rotas somente após o usuário se autenticar.

4. PADRÕES DE PROJETO

Nesta API usaremos alguns padrões simples que ajudam a deixar o código mais organizado e fácil de manter, como: I) DTO (Data Transfer Object) - Serve para organizar os dados que entram e saem da API. Por exemplo, ao enviar uma reserva, usamos um objeto com apenas os dados necessários, como data e horário, sem enviar a sala inteira. II) Repository - Esse padrão é usado para separar a parte do código que fala com o banco de dados. Por exemplo, temos um ReservaRepository que cuida apenas de salvar, editar e buscar reservas no banco.

Esses padrões ajudam a deixar o projeto mais limpo e separado por responsabilidades.

5. FRAMEWORKS

5.1 Spring Framework

O **Spring Boot** ajuda a criar APIs de forma rápida e prática. Ele já vem com várias ferramentas prontas que usaremos no projeto:

- **Spring Web:** para criar os caminhos da API (rotas REST)
- **Spring Security:** para proteger as rotas com login
- **Spring Data JPA:** para facilitar a conexão com o banco de dados

Tudo isso permite focar mais nas regras do sistema e menos em configurações complicadas.

5.2 Hibernate

O **Hibernate** é uma ferramenta que facilita salvar e buscar dados no banco de dados usando objetos Java. Em vez de escrever comandos SQL, usamos classes e ele faz o trabalho por trás.

Por exemplo, ao criar uma classe Reserva, o Hibernate cuida de gravar e buscar os dados dela nas tabelas do banco.

6. REFERÊNCIAS

SPRING.IO. *Spring Boot Documentation*. Disponível em: https://docs.spring.io/spring-boot/docs/current/reference/html/.

SPRING.IO. *Spring Security Reference*. Disponível em: https://docs.spring.io/spring-security/reference/.

HIBERNATE. *Hibernate ORM Documentation*. Disponível em: https://hibernate.org/orm/documentation/.

ORACLE. *Java Platform, Enterprise Edition (Java EE)*. Disponível em: https://docs.oracle.com/javaee/.