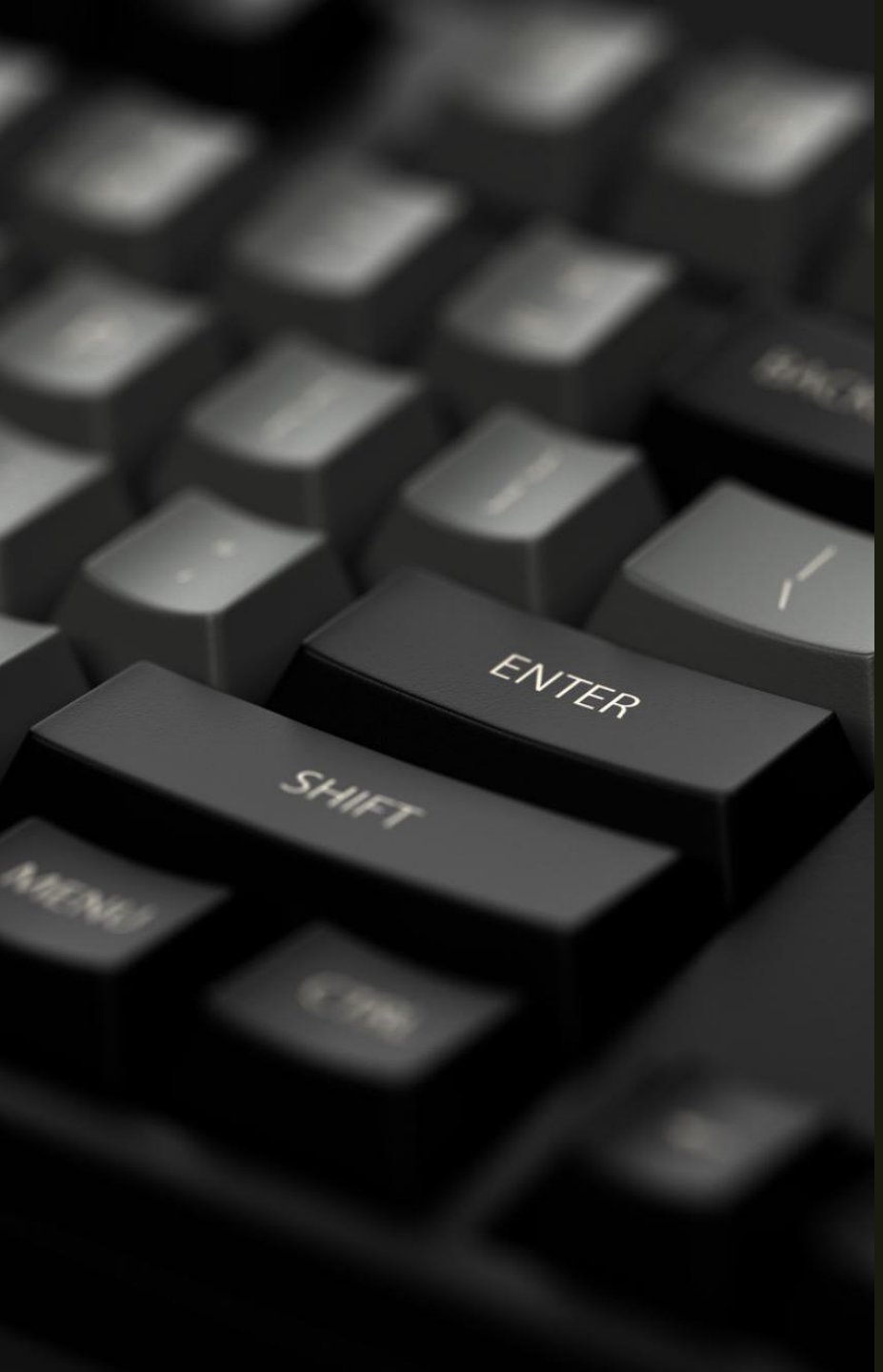
SISTEMA DE ALUGUEL DE SALAS

Lucas Franco Zanforlim



Introdução e Objetivo do Sistema

- **Contexto do problema:** O controle de reservas ainda é feito manualmente em muitos locais, usando planilhas ou mensagens. Isso causa erros, falta de organização, dificuldade para ver a disponibilidade e conflitos de horários.
- **Motivação:** Criar uma solução moderna que automatize o agendamento, melhore a gestão de salas, reduza erros e ofereça mais praticidade e transparência para administradores e usuários.
- **Público-Alvo:** Empresas, escolas, universidades, organizações, e qualquer outro usuário que precise visualizar e reservar salas de formas simples e rápida.

Principais Histórias de Usuário

1- Como usuário, quero visualizar a lista de salas disponíveis para escolher rapidamente qual sala atende minha necessidade

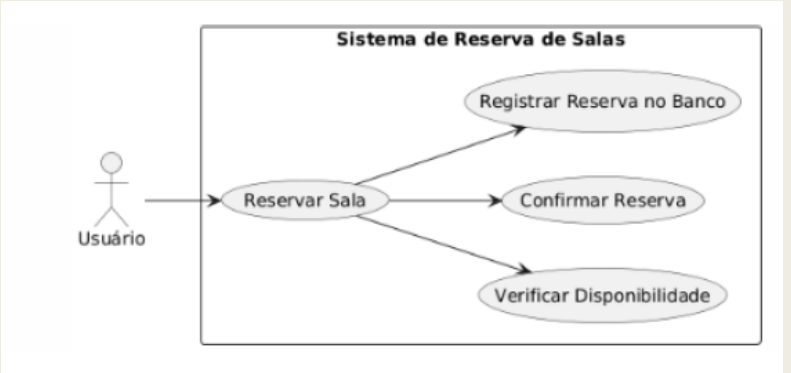
2- Como usuário, quero fazer uma reserva de sala para garantir o uso do espaço no horário desejado

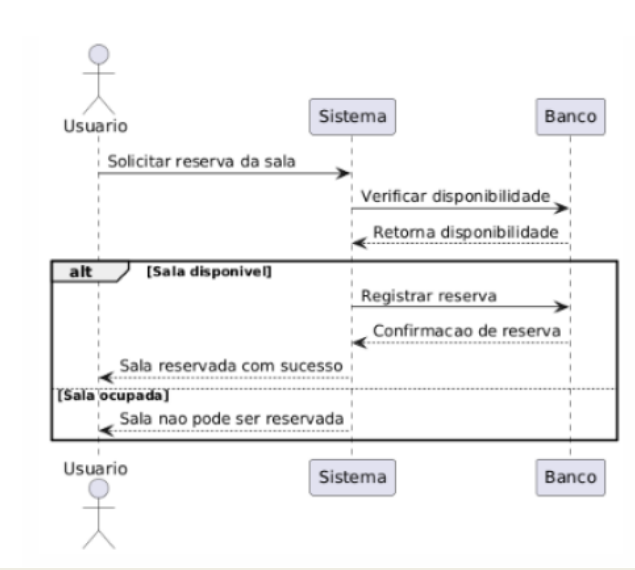
3- Como administrador, quero cadastrar, editar e remover salas para manter o sistema sempre atualizado.

4- Como administrador, quer visualizar todas as reservas para acompanhar, organizar e evitar conflitos de horários

Diagramas utilizados

- Diagrama de casos de uso
- Diagrama de sequência





Arquitetura do Sistema

Padrões Usados:

API fina: controladores com
regras isRoomAvailable/overlaps, validação no servidor.

Dados centralizados: query() com pool + placeholders;
front desacoplado consumindo JSON.

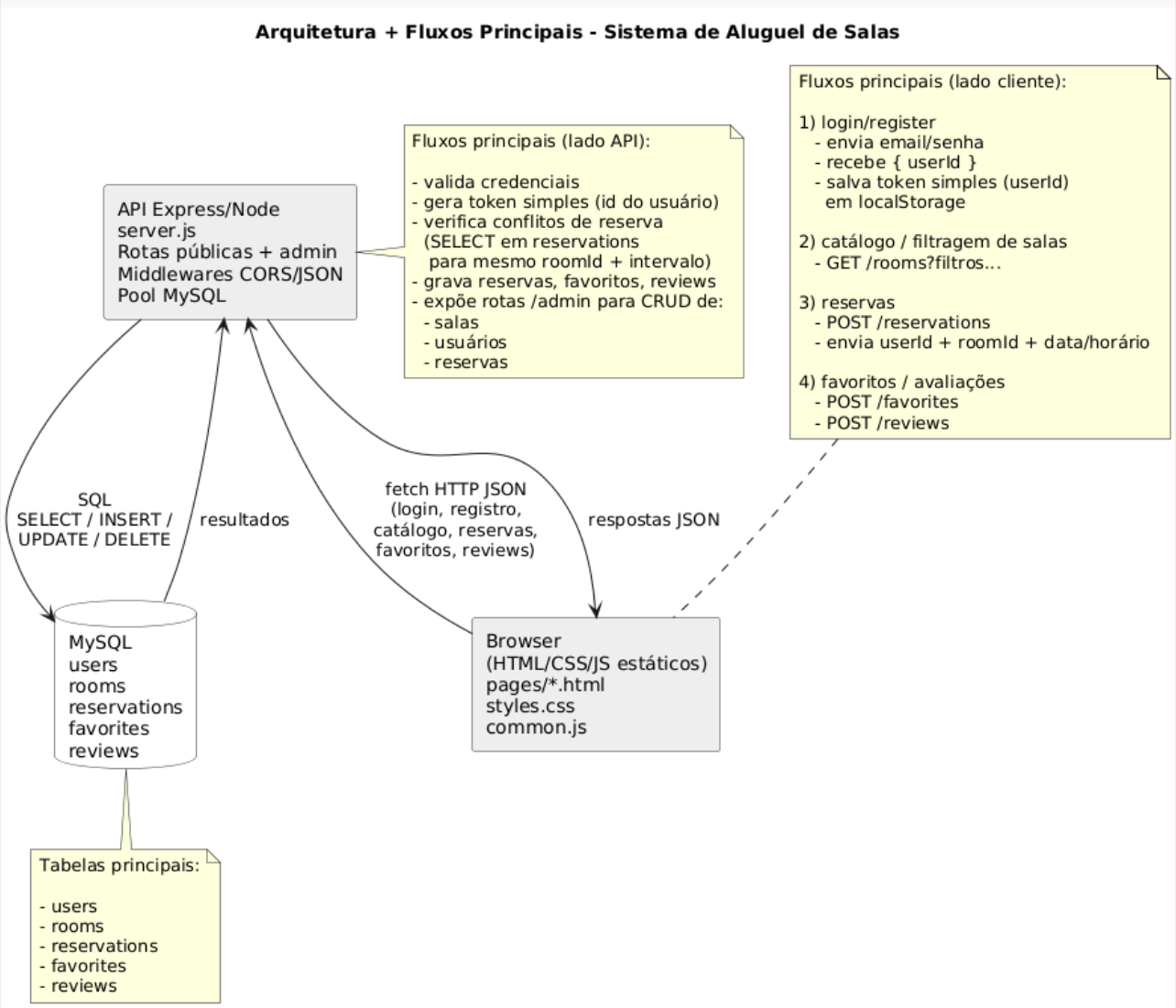
REST pragmático: GET/POST/PUT/DELETE em /rooms, /reservations, /favorites, /reviews, /admin/*.

Justificativas:

Entrega rápida e simples: Express + MySQL; front/back bem separados para trocar UI depois.

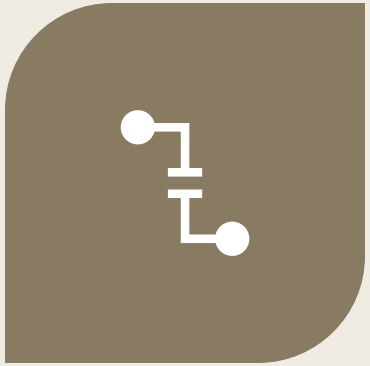
Segurança/performance básicas: pool + parâmetros; regra de disponibilidade garantida no backend.

Governança leve: rotas admin com isAdmin; sessão simples (token=id em localStorage) adequada para POC.



Implementação

COMPONENTES: FRONT ESTÁTICO (HTML/CSS/JS) → API EXPRESS (SERVER.JS, ROTAS PÚBLICAS/ADMIN) → MYSQL (USERS, ROOMS, RESERVATIONS, FAVORITES, REVIEWS).

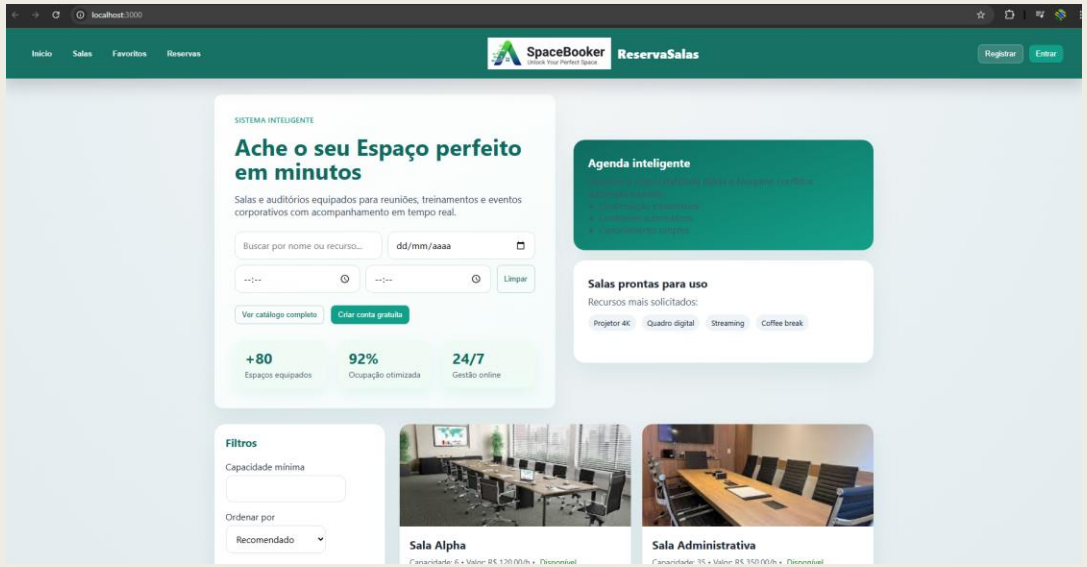


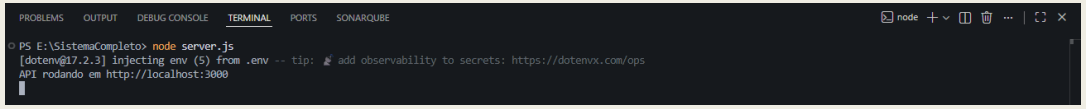
TECNOLOGIAS: NODE 18+, EXPRESS + CORS/JSON, MYSQL2/PROMISE (POOL), FETCH VANILLA, DOTENV.

FLUXO: HTTP → CONTROLADOR EXPRESS → REGRAS (ISROOMAVAILABLE, OVERLAPS, ISADMIN) → QUERY() → SQL PARAMETRIZADO → JSON.

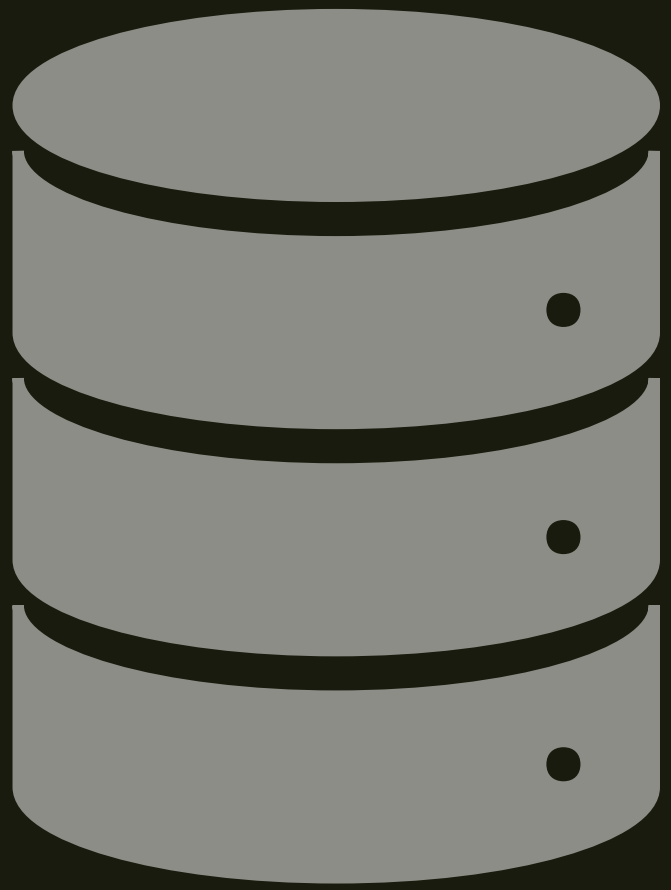
Testes e Qualidade

- Estratégia de testes: foco em API end-to-end contra servidor local (BASE_URL), cobrindo validações e códigos de erro (ex.: 400/404) em auth, rooms e reservations; orientação a casos mínimos de regressão rápida.
- Ferramentas: node:test + assert nativos do Node 18 e fetch para chamar a API; execução direta com node testes/*.test.js apontando para o servidor rodando.





DEMONSTRAÇÃO DA APLICAÇÃO



Conclusões

- Principais aprendizados: checar regras no backend (disponibilidade), usar pool/queries parametrizadas, manter o front só consumindo JSON.
- Melhorias futuras: login/token de verdade, separar regras de negócio em serviços, mais testes e uma UI mais moderna.
- O que faríamos diferente: alinhar nomes e domínio desde o começo, definir autenticação antes, criar testes e dados de exemplo logo no início.