Relatório de Arquitetura e Design: Projeto "Seu Cantinho"

Márcio Vinícius de Carvalho Marinho GRR20204089

30 de novembro de 2025

1 Arquitetura Proposta

Para garantir que o sistema seja robusto, testável e fácil de manter a longo prazo, adotamos uma abordagem baseada na Arquitetura Hexagonal, organizada através de uma estrutura de Pacotes por Funcionalidade.

1.1 Por que Hexagonal?

A Arquitetura Hexagonal isola o negócio de dependências externas (Banco de Dados, API Web, UI). Isso melhora consideravelmente a testabilidade, manutenibilidade e escalabilidade do código, conforme novas funcionalidades e novas regras precisam ser adicionadas. Não é um projeto tão grande que seja necessário utilizar microsserviços, mas como já está em um processo de expansão, o hexagonal possui capacidade de atender essa demanda de escalabilidade, enquanto não aumenta drasticamente a complexidade e custo de desenvolvimento e manutenção.

2 Tecnologias Utilizadas

O sistema foi desenvolvido com Java 17 + Spring Boot 3.x no backend, API REST documentada via Swagger. O frontend Angular 17 + TypeScript. O banco de dados é PostgreSQL 15 com Spring Data JPA, e todo o ambiente é containerizado com Docker Compose. Testes unitários foram feitos com Jest/JUnit/Mockito e JWT para autenticação stateless.

2.1 Prevenção de Double Booking

O sistema garante que um espaço não seja reservado duas vezes para a mesma data através de uma constraint definida no banco de dados PostgreSQL. A linha `CREATE UNIQUE INDEX idx_reserva_unica_ativa ON tb_reserva (espaco_id, data_evento) WHERE status NOT IN ('CANCELADA', 'FINALIZADA')` cria um índice que bloqueia inserções duplicadas apenas para reservas ativas, permitindo que um espaço seja usado no mesmo dia após a reserva ser cancelada ou finalizada. O banco de dados garante atomicidade mesmo em cenários de concorrência. O código Java também realiza uma validação prévia, melhorando a experiência do usuário ao retornar uma mensagem de erro amigável antes de tentar a inserção no banco.

2.2 Organização do Projeto

O sistema está dividido em três grandes blocos lógicos para cada funcionalidade (ex: Cliente, Reserva, Espaço):

- **Domain** Responsável pelas entidades (ex: `Cliente`, `Reserva`) e as regras de negócio de forma geral.
- **Application** Camada que coordena as ações. Ela recebe um pedido de fora, chama o domínio para processar e devolve o resultado.
- **Infrastructure** Aqui ficam os Controladores REST que se comunincam com o Angular, e os Repositórios JPA que se comunicam com o Banco de Dados.

3 Mapeamento UML → Código

Uma das maiores dificuldades foi manter o código fiel ao mapeamento UML. Foi necessário alterar diversas vezes os diagramas, com base nas novas decisões que iam surgindo durante o desenvolvimento.

3.1 Estrutura de Pastas

A estrutura do backend segue o diagrama de componentes. Se você abrir a pasta `backend/src/main/java/c` encontrará:

- `domain/`: Contém a classe `Cliente.java` (Entidade) e `port/` (Interfaces que definem como salvar/buscar dados).
- `application/service/`: Contém `ClienteService.java`, que implementa os casos de uso desenhados.
- `infrastructure/adapter/in/web/`: Contém `ClienteWebAdapter.java` (o Controller), que é a porta de entrada da API.

3.2 Diagrama de Classes vs. Entidades

O diagrama de classes (`classDiagram.mmd`) definiu que `Usuario` seria a base para `Cliente`, `Funcionario` e `Administrador`. No código, isso foi implementado usando herança e composição, garantindo que regras comuns (como login e senha) fiquem centralizadas, enquanto dados específicos (como CPF do cliente ou Matrícula do funcionário) fiquem separados.

3.3 Máquina de Estados

O ciclo de vida da reserva (conforme o diagrama stateDiagram.mmd) — de Aguardando Sinal até Quitada ou Cancelada — foi implementado utilizando o State Design Pattern para controlar as transições de estado válidas. A determinação de qual novo estado deve ser assumido, após um pagamento, é feita através do Strategy Design Pattern, que seleciona a estratégia apropriada baseada no tipo de pagamento (SINAL, QUITACAO ou TOTAL).

4 Decisões de Design e Trade-offs

4.1 Arquitetura Hexagonal vs. Simplicidade

Decisão: Adotar Hexagonal.
Ganho: Isolamento das camadas através de portas e adaptadores. Mais testabilidade, escalabilidade e melhor manutenibilidade

Custo: Mais arquivos e diretórios para criar.

4.2 Hexagonal Puro vs. Não Puro

Decisão: Manter não puro. O Domain não foi isolado do Spring Framework
Ganho: Simplifica a configuração já que o Spring gerencia automaticamente a injeção de dependências.
Custo: Acopla o Domain a uma tecnologia externa, violando os princípios da Arquitetura Hexagonal onde o domínio deveria ser completamente isolado do framework.

4.3 Monorepo (Backend + Frontend juntos)

Decisão: Manter tudo no mesmo repositório Git.
Ganho: Facilita a visão do projeto como um todo e simplifica o processo de build e deploy em ambiente de desenvolvimento.
Custo: O repositório fica maior e mistura linguagens java e TypeScript.

4.4 Uso de DTOs (Data Transfer Objects)

Decisão: Nunca expor as Entidades do banco diretamente para a API.
Ganho: Segurança e desacoplamento. Podemos mudar a estrutura do banco sem quebrar o Frontend.
Custo: Necessidade de criar classes "espelho"(DTOs) e conversores (Mappers), o que gera código repetitivo.

5 Instruções de Execução

Para ver o sistema funcionando, preparamos um ambiente baseado em Docker, que elimina a famosa frase "na minha máquina funciona".

5.1 Pré-requisitos

- Docker e Docker Compose instalados.
- Portas 5433 (Banco), 8080 (API) e 8081 (Frontend) livres.

5.2 Passo a Passo

1. Abra o terminal na pasta raiz do projeto.
2. Execute o comando:

```
docker-compose up -d
```

3. Aguarde alguns minutos enquanto o Docker baixa as imagens e compila o projeto.
4. Acesse no navegador:
   - **Frontend:** `http://localhost:8081`
   - **Swagger API:** `http://localhost:8080/swagger-ui.html`

5.3 Usuários para Teste

O sistema já vem com dados populados. A senha para todos é `password123`.

- **Admin:** admin@seucantinho.com
- **Funcionário:** joao.pr@seucantinho.com
- **Funcionário:** ana.sc@seucantinho.com
- **Cliente:** cliente1@gmail.com
- **Cliente:** cliente2@gmail.com

6 Funcionalidades do Sistema

6.1 Gestão de Usuários e Autenticação

- **Login e Controle de Acesso**: Sistema de autenticação baseado em JWT (JSON Web Token) com três perfis distintos: Administrador, Funcionário e Cliente.
- **Cadastro de Clientes**: Novos clientes podem se registrar.
- **Gerenciamento de Funcionários**: Administradores podem cadastrar funcionários vinculados a filiais específicas.

6.2 Gestão de Filiais e Espaços

- **Cadastro de Filiais**: Registro de unidades em diferentes cidades, com endereço e telefone de contato.
- **Catálogo de Espaços**: Cada filial possui espaços cadastrados com nome, descrição, capacidade máxima de pessoas, preço da diária e foto principal.
- **Consulta**: Clientes podem visualizar os espaços disponíveis, suas características e valores, sem necessidade de login.

6.3 Sistema de Reservas

- **Criar Reserva**: Cliente autenticado seleciona um espaço, escolhe a data do evento e adiciona observações personalizadas.
- **Verificação de Disponibilidade**: O sistema bloqueia automaticamente reservas duplicadas — apenas uma reserva ativa por espaço/data é permitida.
- **Controle de Status**: Cada reserva possui um ciclo de vida bem definido:
  1. *Aguardando Sinal* → Cliente acabou de criar a reserva
  2. *Confirmada* → Sinal foi pago (50% do valor)
  3. *Quitada* → Valor total foi pago
  4. *Finalizada* → Evento ocorreu e a reserva foi encerrada
  5. *Cancelada* → Reserva cancelada
- **Histórico de Reservas**: Clientes visualizam suas reservas passadas e futuras. Funcionários veem todas as reservas de sua filial.

6.4 Sistema de Pagamentos

- **Pagamento de Sinal**: Cliente paga uma parte do valor para confirmar a reserva. O status muda automaticamente para *Confirmada*.
- **Quitação**: Pagamento do restante do valor, alterando o status para *Quitada*.
- **Pagamento Total**: Opção de pagar 100% de uma vez, indo direto para *Quitada*.
- **Histórico de Transações**: Registro completo de todos os pagamentos.

6.5 Painel Administrativo

- **Dashboard**: Visão geral com métricas de reservas, faturamento e ocupação dos espaços.
- **Gestão**: Administradores têm acesso para criar, editar ou desativar filiais, espaços, funcionários e visualizar todas as reservas do sistema.