Relatório Final do Projeto

Sistema de Reserva de Hotéis

Disciplina: Projeto e Modelagem de Banco de Dados
Professora: Gabrielle K. Canalle
Ano: 2025
Grupo: Gabriel Antonio, Gustavo Laporte, Matheos Guerra, Pedro Dhalia

---

1. Introdução

Este relatório tem como objetivo documentar todas as etapas do desenvolvimento do sistema de **Reserva de Hotéis**, realizado como projeto final da disciplina de **Projeto e Modelagem de Banco de Dados**. O sistema foi concebido para oferecer uma solução completa de gestão hoteleira, com funcionalidades para gerenciar reservas, pagamentos, manutenções, serviços, avaliações, relações funcionais e muito mais.

O desenvolvimento foi dividido em dois módulos:

- **Módulo 1:** Modelagem, scripts SQL e primeira interface
- **Módulo 2:** CRUDs completos com backend em Spring Boot, frontend React, dashboard e consultas avançadas

---

2. Etapas Realizadas

Módulo 1:

- Levantamento do mini-mundo
- Elaboração do Modelo Conceitual (MER)
- Geração do Modelo Lógico Relacional
- Criação dos scripts SQL de criação e inserção de dados
- Primeira versão de CRUD em Java com JDBC

Módulo 2:

- Refatoramento do backend utilizando Java com Spring Boot
- Construção do frontend com React + Vite
- Integração via API REST com JSON
- Criação de testes de todas as rotas
- Desenvolvimento de um dashboard com dados visuais
- Execução de consultas SQL avançadas

---

3. Modelo Conceitual

O MER foi desenvolvido com o BRModelo e contempla os seguintes conceitos:

- Atributos compostos e multivalorados (telefone)
- Entidades fracas (Avaliação)
- Herança (Pessoa como supertipo de Funcionário e Hóspede)
- Auto-relacionamento (Supervisiona)
- Relacionamentos ternários (Executa: relaciona Funcionario, Manutenção e Quarto)

Imagem: `docs/modelo_conceitual.png`

---

4. Modelo Lógico Relacional

O modelo lógico foi elaborado a partir do MER e adaptado para o SGBD MySQL. Foram definidos:

- Tipos de dados apropriados
- Restrições de integridade (PK, FK, UNIQUE, CHECK)
- Chaves compostas em relacionamentos (Executa, Supervisiona, Possui)

Arquivo: `docs/modelo_logico.md`

---

5. Scripts SQL

script_criacao.sql:

Contém a criação de todas as tabelas relacionais. Respeita a modelagem com:

- Chaves estrangeiras
- Constraints de unicidade
- Chaves compostas

script_insercao.sql:

Contém dados realistas para todas as tabelas.
Inclui:

- Diversas pessoas, com CPF, data de nascimento e endereço
- Quarto com diferentes tipos, preços e status
- Reservas com serviços, pagamentos, avaliações e manutenções

---

6. Backend (Java + Spring Boot)

Estrutura:

- `model/`: entidades JPA com mapeamento direto ao banco
- `repository/`: interfaces que extendem JpaRepository
- `controller/`: endpoints RESTful para cada entidade

Funcionalidades:

- CRUD completo para todas as 13 tabelas

- Integração com banco via Spring Data JPA
- Testes de todas as rotas via Postman/axios
- Tratamento de CORS e JSON

Execução:

cd backend
./mvnw spring-boot:run

---

7. Frontend (React + Vite)

Estrutura:

- `components/`: formulários e listas de cada entidade
- `pages/`: rotas para as views do sistema
- `api/`: funções de comunicação com backend via axios

Funcionalidades:

- Interface visual responsiva
- Telas para cada entidade com operações CRUD
- Integração com backend Java

Execução:

cd frontend
npm install
npm run dev

---

8. Dashboard de Visualização de Dados

- Criado na interface em React

- Apresenta gráficos como:
    - Quartos ocupados vs livres
    - Forma de pagamento mais utilizada
    - Notas de avaliação
    - Reservas por tipo de quarto

---

9. Consultas SQL Avançadas

Arquivo: `sql/consultas_avancadas.sql`
Inclui:

- Junções entre 3+ tabelas
- Filtros condicionais
- Agrupamentos com GROUP BY e HAVING
- Subconsultas correlacionadas

Exemplos:

- Quartos com manutenções recentes
- Reservas com serviços acima de R$50
- Funcionários que supervisionam alguém

---

10. Testes de Funcionalidades

Todos os endpoints da API REST foram testados manualmente utilizando o Postman, verificando o correto funcionamento das operações de criação, listagem, atualização e exclusão de dados em todas as entidades do sistema.

Exemplos de rotas testadas:

- `GET /api/pessoas` – listar pessoas cadastradas

- `POST /api/quartos` – cadastrar novo quarto
- `PUT /api/reservas/1` – atualizar reserva existente
- `DELETE /api/avaliacoes/3` – deletar uma avaliação

Esses testes garantiram que o sistema esteja funcional em sua totalidade e que os relacionamentos entre tabelas estejam sendo respeitados conforme o modelo lógico relacional.

---

11. Estrutura do Projeto (Resumida)

```
hotel-reserva-sistema-bd/
├── backend/ (Java + Spring Boot)
│   ├── controller/
│   ├── model/
│   ├── repository/
├── frontend/ (React + Vite)
│   ├── components/
│   ├── pages/
│   ├── api/
├── sql/
│   ├── script_criacao.sql
│   ├── script_insercao.sql
│   └── consultas_avancadas.sql
├── docs/
│   ├── modelo_conceitual.png
│   ├── modelo_logico.md
│   └── mini_mundo.md
└── README.md
```

---

12. Conclusão

Esse projeto foi, sem dúvida, um dos mais desafiadores que enfrentamos na graduação até agora. Desde o início, sabíamos que não seria simples — começamos enfrentando dificuldades já na modelagem conceitual. Apanhamos para estruturar o MER, entender os relacionamentos entre as entidades, e principalmente para representar de forma correta os casos de herança e relacionamentos N:N. A transição do modelo conceitual para o modelo lógico também exigiu atenção redobrada, especialmente na definição de chaves estrangeiras e integridade referencial.

Na parte de implementação, tomamos inicialmente o caminho mais conhecido: construímos todo o backend com Node.js e MySQL, utilizando o pacote mysql2 com Promises. Estava tudo funcionando — endpoints criados, testados, banco conectado. Mas ao avaliarmos os critérios da disciplina e alinharmos com os requisitos da professora, percebemos que o mais adequado seria adotar Java com Spring Boot. E foi aí que começou a segunda fase da jornada: refatoramos toda a API, recriamos os CRUDs em Java, configuramos o Spring Data JPA, lidamos com as entidades, repositórios, controllers, serviços e com a integração com o banco de dados relacional. Foi trabalhoso, mas o aprendizado foi enorme.

Enquanto isso, o frontend também evoluía. A equipe trabalhou usando React com Vite para construir uma interface moderna, funcional e que conseguisse consumir os dados da API com clareza. Integramos rotas, componentes reutilizáveis, estilização e até um dashboard interativo que visualiza os dados diretamente do banco — o que exigiu melhorar o script de inserção para tornar a visualização mais interessante.

Ao final do processo, ver o sistema funcionando de ponta a ponta — banco de dados populado, backend com API REST estruturada e testada, e um frontend conectado com tudo — foi uma sensação de missão cumprida. Mais do que cumprir requisitos acadêmicos, a gente construiu algo do zero, em equipe, e vimos ele ganhar vida.

Além do conhecimento técnico, esse projeto ensinou muito sobre organização, versionamento com GitHub, divisão de tarefas, paciência e, principalmente, trabalho em equipe. Cada um contribuiu com dedicação em sua parte, e o resultado foi um sistema funcional, coerente e entregue com muito orgulho.

Agora é hora de respirar fundo e celebrar: projeto final entregue com sucesso — e com a certeza de que saímos desse desafio mais preparados para o mundo real do desenvolvimento.