Documentação Técnica — RoomBooking

1. Descrição da ideia e do negócio

O RoomBooking resolve o problema de gestão de reservas de salas em ambientes corporativos e acadêmicos, evitando conflitos de horários e excesso de lotação. A solução centraliza o controle de reservas, permitindo que usuários consultem, criem e gerenciem reservas de forma eficiente, com validações automáticas e notificações integradas.

---

2. Regras de negócio implementadas

Salas

- **Criação de sala:** Nome obrigatório, capacidade entre 1 e 100 pessoas.
- **Atualização de sala:** Permite alterar nome e capacidade, respeitando as mesmas regras de criação.
- **Validações:** Nome não pode ser vazio; capacidade não pode ser nula, negativa ou acima de 100.

Reservas

- **Criação de reserva:**
  - Não permite reservas com datas no passado.
  - Data de início deve ser anterior à de término.
  - Organizador obrigatório, até 100 caracteres.
  - Não permite sobreposição de horários para a mesma sala.
  - Não permite exceder a capacidade da sala.
- **Validações:** Todas as regras acima são validadas via FluentValidation e exceções de domínio.

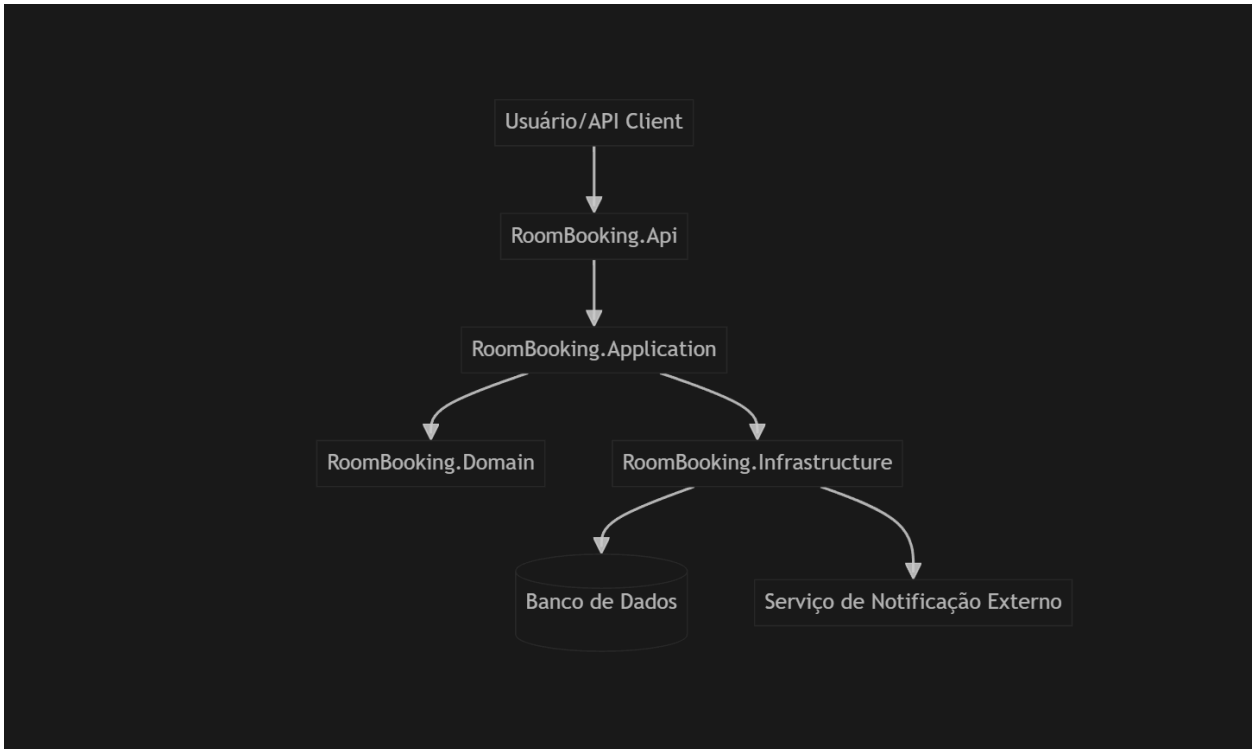
---

3. Estrutura da solução

- **RoomBooking.Domain:** Entidades, value objects, interfaces e exceções de domínio.
- **RoomBooking.Application:** Comandos, queries, DTOs, eventos, validações, behaviors e serviços de aplicação.
- **RoomBooking.Infrastructure:** Persistência (EF Core), repositórios, integrações externas (HttpClient, Polly), migrations.
- **RoomBooking.Api:** Controllers, middlewares, configuração de DI, inicialização da aplicação.
- **RoomBooking.Tests:** Testes unitários e de integração.

---

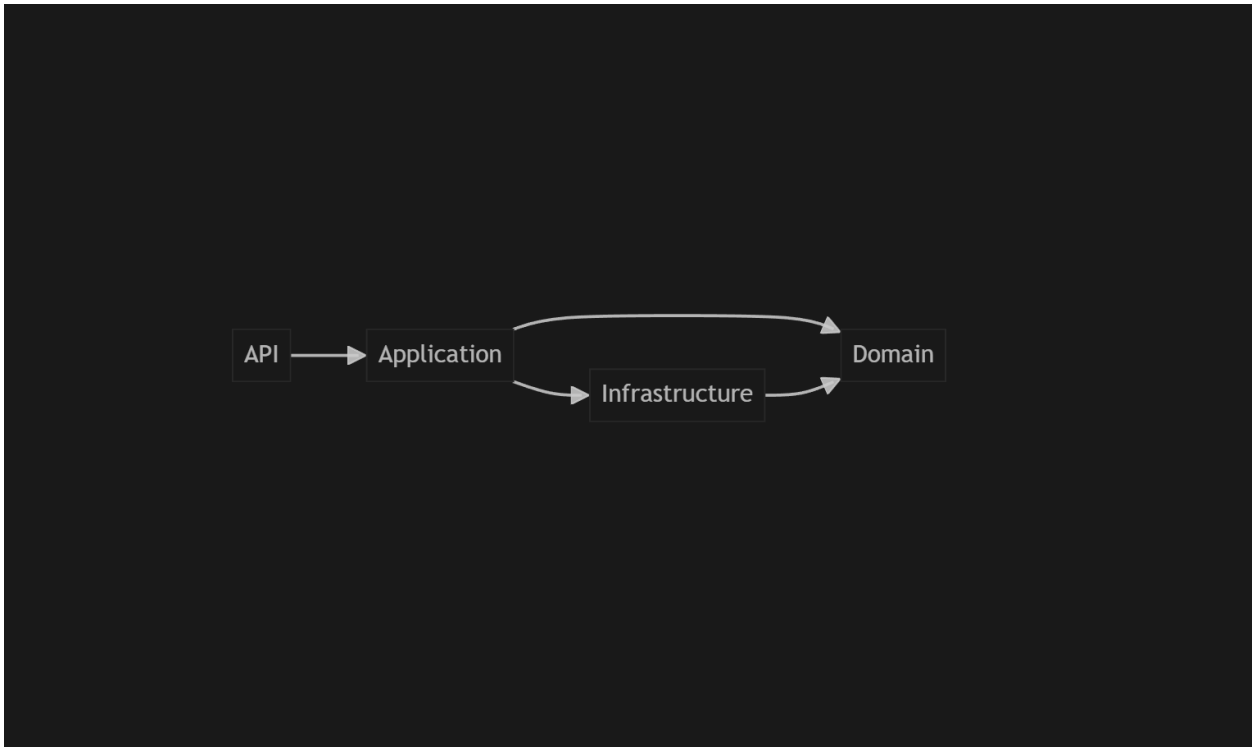
4. Diagrama da arquitetura



Descrição:
A arquitetura segue o padrão Clean Architecture, separando responsabilidades em camadas independentes. A API recebe as requisições, delega para a camada de aplicação, que orquestra regras de negócio (Domain) e persistência/integracão (Infrastructure).

---

5. Diagrama das camadas



- **API:** Controllers, middlewares, configuração.
- **Application:** CQRS, validações, behaviors, serviços de aplicação.
- **Domain:** Entidades, regras de negócio, value objects, interfaces.
- **Infrastructure:** Persistência, repositórios, integrações externas.

---

6. Pontos técnicos e padrões utilizados

- **DDD (Domain-Driven Design):**
  Entidades e value objects modelam o domínio. Exceções de domínio garantem integridade das regras.

- **SOLID:**
  - *Single Responsibility:* Cada classe tem uma responsabilidade clara.
  - *Open/Closed:* Validações e handlers são facilmente estendidos.
  - *Dependency Inversion:* Interfaces e injeção de dependência em todas as camadas.
- **CQRS:**
  Separação entre comandos (escrita) e queries (leitura) usando MediatR.

- **Mediator (MediatR):**
  Desacopla a orquestração de comandos/queries/eventos.

- **FluentValidation:**
  Validação declarativa e reutilizável para comandos.
- **AutoMapper:**
  Mapeamento automático entre entidades e DTOs.
- **Polly:**
  Resiliência em integrações externas (retry, circuit breaker) no envio de notificações.
- **Middlewares customizados:**
  Tratamento global de exceções, respostas padronizadas de erro.
- **Testes automatizados:**
  Cobertura de regras de negócio e validações.