## **Documentação da Arquitetura - AgendaPro Web API**

### **1. Visão Geral e Filosofia de Design**

O projeto **AgendaPro API** serve como o backend para um sistema de controlo de reservas de salas de reunião. A sua principal responsabilidade é gerir a lógica de negócio, as regras de validação e a persistência de dados para as entidades principais: Salas, Usuarios e Reservas.

A arquitetura foi concebida com base nos princípios da **Arquitetura Limpa (Clean Architecture)**, fortemente influenciada pelo **Domain-Driven Design (DDD)**. O objetivo central é criar um sistema:

* **Desacoplado:** As camadas são independentes. A lógica de negócio não depende de detalhes de infraestrutura (como o banco de dados) ou de apresentação (a API).
* **Testável:** Cada camada, especialmente a de Aplicação e a de Domínio, pode ser testada de forma isolada, garantindo a fiabilidade do sistema.
* **Escalável e de Fácil Manutenção:** A separação clara de responsabilidades permite que novas funcionalidades sejam adicionadas com impacto mínimo nas partes existentes do sistema.

O fluxo de dependências é estritamente unidirecional, sempre a apontar para o núcleo do sistema (o Domínio).

[ Camada de Apresentação (API) ] -> [ Camada de Aplicação ] -> [ Camada de Domínio ]  
 ^  
 |  
 [ Camada de Infraestrutura ] -----´

### **2. Estrutura das Camadas**

A solução está dividida em quatro projetos principais, cada um representando uma camada lógica da arquitetura.

#### **2.1. AgendaPro.Dominio (O Coração)**

Esta é a camada mais interna e importante. Contém as regras de negócio mais puras e essenciais do sistema. Não possui dependências com nenhuma outra camada.

* **Componentes:**
  + **Entidades**: Classes que representam os conceitos de negócio (Reserva, Sala, Usuario). Elas contêm não apenas os dados (propriedades), mas também os métodos que validam e manipulam o seu próprio estado (ex: o método Reserva.Cancelar() contém a lógica de cancelamento das 24 horas).
  + **Enums**: Tipos enumerados que representam estados definidos do negócio (StatusReserva).
  + **Interfaces de Repositório**: Contratos (IReservaRepositorio, ISalaRepositorio, etc.) que definem como a camada de Domínio espera interagir com a persistência de dados, sem conhecer os detalhes da implementação (padrão **Repository**). A interface IUnitOfWork define o contrato para garantir a atomicidade das transações.

#### **2.2. AgendaPro.Aplicacao (O Cérebro)**

Esta camada orquestra os casos de uso (use cases) do sistema. Ela atua como um intermediário entre a camada de apresentação e o núcleo de negócio.

* **Componentes:**
  + **Commands e Queries (CQRS)**: Em vez de serviços genéricos, cada ação é modelada como uma mensagem distinta.
    - **Commands** (CriarSalaComando, EditarUsuarioComando): Representam uma intenção de alterar o estado do sistema.
    - **Queries** (ListarSalasQuery, RetornoSalaIdComando): Representam uma intenção de ler dados, sem os modificar.
  + **Handlers**: Classes que implementam a lógica para um Command ou Query específico (ex: CriarReservaComandoHandler). É aqui que a orquestração acontece: o handler recebe a mensagem, usa os repositórios para interagir com os dados e chama os métodos de negócio das entidades.
  + **DTOs (Data Transfer Objects)**: Classes simples (SalaDto, UsuarioDto) que definem o formato dos dados que a camada de Aplicação devolve para o exterior. Isto evita expor as entidades de domínio diretamente.
  + **Padrão Mediator**: Usamos a biblioteca MediatR para implementar o padrão Mediator. Isto desacopla quem envia o pedido (o Controller) de quem o executa (o Handler), tornando o código dos controllers extremamente limpo.

#### **2.3. AgendaPro.Infraestrutura (Os Braços e Pernas)**

Esta camada contém as implementações técnicas de tudo o que é externo à aplicação. Ela implementa as interfaces definidas nas camadas de Aplicação e Domínio.

* **Componentes:**
  + **Persistence**: Contém tudo relacionado à persistência de dados.
    - **DbContext**: A implementação da sessão com o banco de dados usando o **Entity Framework Core**.
    - **Repositories**: A implementação concreta das interfaces de repositório (ReservaRepository, etc.), que usam o DbContext para executar as queries no banco de dados.
    - **UnitOfWork**: A implementação que chama o SaveChangesAsync() do DbContext.
  + **Services**: Futuramente, conteria a implementação de serviços externos, como o EmailService.

#### **2.4. AgendaPro.API (A Vitrine)**

A camada mais externa, responsável por expor a funcionalidade do sistema ao mundo através do protocolo HTTP.

* **Componentes:**
  + **Controllers**: Pontos de entrada da API. A sua única responsabilidade é receber a requisição HTTP, traduzi-la para um Command ou Query, enviá-la para o MediatR, e formatar a resposta. Eles são propositadamente "magros" e sem lógica de negócio.
  + **Request DTOs**: Classes que modelam o corpo das requisições de entrada (CriarSalaRequest).
  + **Configuração (Program.cs)**: Onde a **Injeção de Dependência** é configurada, "ligando" as interfaces aos seus serviços concretos.

### **3. Fluxo de uma Requisição: Exemplo de Criar uma Sala**

1. Uma requisição POST chega ao endpoint /api/salas com os dados da nova sala no corpo.
2. O SalasController na camada de **API** recebe a requisição.
3. O Controller cria um objeto CriarSalaComando com os dados recebidos.
4. O Controller usa o IMediator para enviar o Command.
5. O MediatR localiza o CriarSalaComandoHandler na camada de **Aplicação** e o invoca.
6. O Handler cria uma nova entidade Sala (da camada de **Domínio**).
7. O Handler usa a interface ISalaRepositorio para adicionar a nova entidade. A injeção de dependência fornece a implementação concreta SalaRepositorio (da camada de **Infraestrutura**).
8. O SalaRepositorio usa o DbContext para adicionar a entidade ao contexto do EF Core.
9. O Handler usa a interface IUnitOfWork para chamar SaveChangesAsync().
10. A alteração é persistida no banco de dados.
11. O Handler retorna o Id da nova sala para o Controller.
12. O Controller retorna uma resposta 201 Created com o novo Id.