🧭 Arquitetura Geral do Projeto TasksApp (Clean Architecture)

Este documento descreve o **fluxo geral de dados e responsabilidades** do projeto TasksApp utilizando **Clean Architecture** com Swift/SwiftUI.

🧱 Camadas da Arquitetura

O projeto está dividido em 4 grandes camadas:

```
[ Presentation ] -> [ Domain ] -> [ Data ] -> [ Armazenamento
(local/mock/api) ]
       ↑               ↓
       |               |
  SwiftUI View     Regras de negócio puras
```

📁 Estrutura Geral

```
TasksApp/
├── Domain/              # Regras de negócio
│   ├── Entities/
│   ├── UseCases/
│   └── Repositories/
├── Data/                # Implementações de repositório e fontes de dados
│   ├── DataSources/
│   └── Repositories/
├── Presentation/        # Views e ViewModels
│   ├── Views/
│   └── ViewModels/
├── DI/                  # Injeção de dependência
└── Tests/               # Testes unitários (não incluído aqui)
```

🔁 Fluxo Geral entre Camadas

1. `SwiftUI View` **(Presentation/Views)** exibe dados observando uma `ViewModel`
2. A `ViewModel` usa os `UseCases` da camada `Domain` para orquestrar ações
3. Cada `UseCase` depende apenas de um **protocolo de repositório**
4. A `camada Data` implementa esses repositórios, acessando dados locais ou remotos
5. O `AppDIContainer` conecta tudo, instanciando dependências

🔍 Conexões Entre Arquivos

📌 View → ViewModel

- `TaskListView` observa `TaskListViewModel`
- Dispara métodos como `addTask`, `deleteTask`, `toggleTask`

📌 ViewModel → UseCases

- `TaskListViewModel` chama:
  - `getTasksUseCase.execute()`
  - `addTaskUseCase.execute(task)`
  - `updateTaskUseCase.execute(task)`
  - `deleteTaskUseCase.execute(task)`

📌 UseCases → Repositório (protocolo)

- Todos os UseCases recebem um `TaskRepository` (protocolo)

📌 Repositório → DataSource

- `TaskRepositoryImpl` implementa o protocolo `TaskRepository`
- Internamente, usa `LocalTaskDataSource` (ou outra fonte de dados)

🔨 Papel de Cada Camada

✅ Presentation

- `TaskListView`: interface visual com SwiftUI

- `TaskListViewModel`: lógica da tela, expõe estado `@Published`

✅ Domain

- `Task.swift`: entidade central da aplicação
- `TaskRepository`: contrato de repositório
- UseCases: encapsulam regras de negócio isoladas

✅ Data

- `LocalTaskDataSource`: fonte de dados local (simulada)
- `TaskRepositoryImpl`: implementação concreta do repositório

✅ DI

- `AppDIContainer`: instância todos os componentes e injeta as dependências corretamente

---

✅ Resumo Final

- O projeto está **100% desacoplado**, favorecendo:
  - Testabilidade
  - Escalabilidade
  - Clareza de responsabilidades
- Cada camada sabe **somente o necessário** sobre a próxima
- O fluxo segue de forma **unidirecional**, conforme os princípios da Clean Architecture