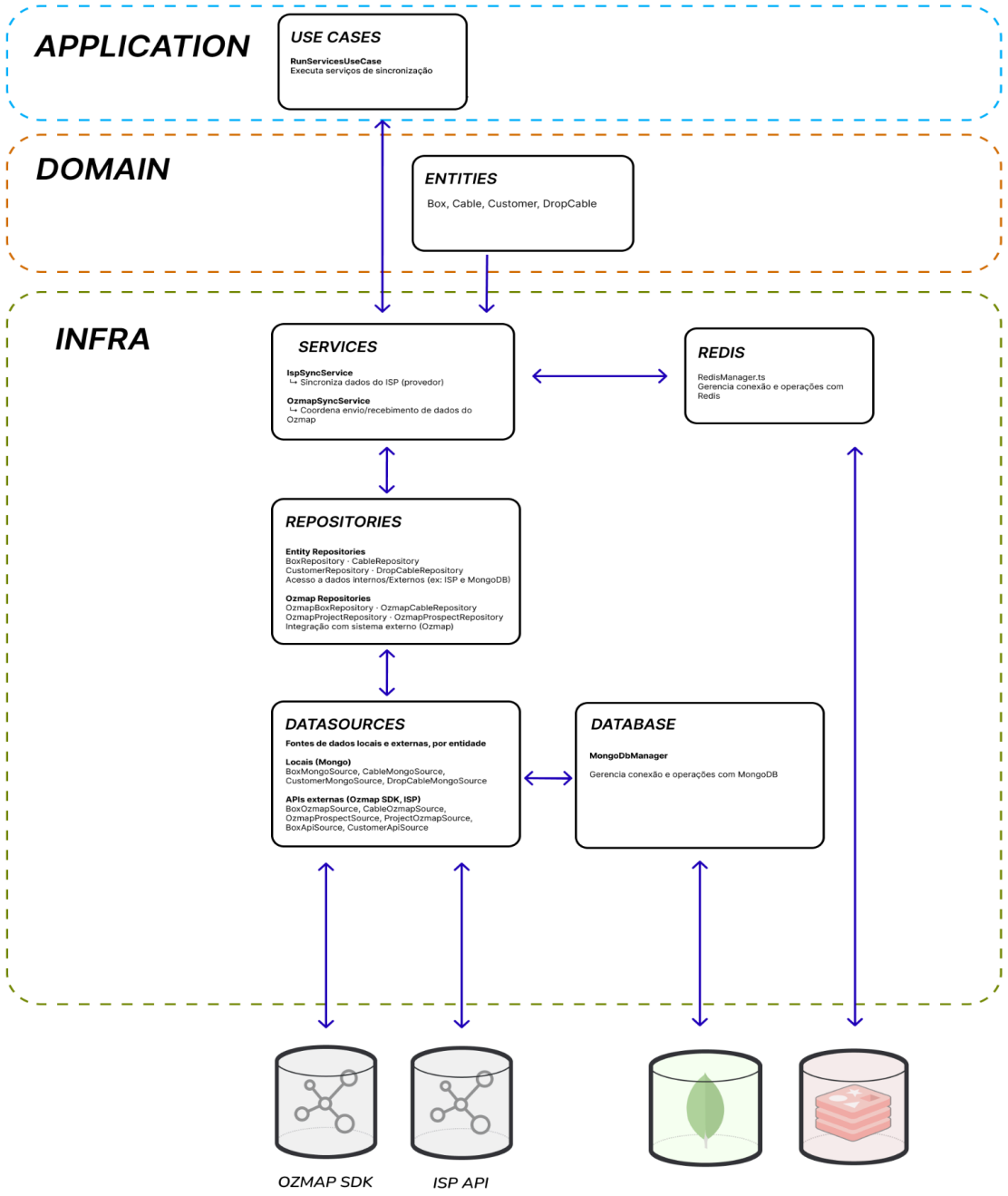
Documentação da Arquitetura do Sistema

Visão Geral

A aplicação é um backend modular desenvolvido para realizar a **sincronização de dados entre fontes locais (como MongoDB e Redis)** e **serviços externos (como Ozmap e ISP)**. Sua estrutura adota os princípios da **Clean Architecture** e **Domain-Driven Design (DDD)**, visando separação de responsabilidades, testabilidade e facilidade de manutenção.



1. Domain

Define as entidades centrais do negócio, representando conceitos puros sem dependências externas.

- **Entidades**:
    - `Box`
    - `Cable`
    - `Customer`
    - `DropCable`
      (Local: `domain/entities`)

---

2. Application

Contém os **casos de uso**, representando regras de negócio da aplicação.

- **Use Case Principal**: `RunServicesUseCase`
    - Coordena os fluxos de sincronização dos serviços.
    - **Ponto de entrada do sistema**, invocado pela `main` (`index.ts`).
      (Local: `application/use-cases`)

---

3. Infrastructure

Lida com as tecnologias externas e fontes de dados, dividida em vários módulos:

a. Datasources

Responsáveis por buscar/gravar dados em fontes externas e locais, organizados por entidade.

- **Locais (MongoDB)**:

    - Ex: `BoxMongoSource`, `CableMongoSource`, `CustomerMongoSource`, `DropCableMongoSource`

- **APIs Externas (Ozmap, ISP)**:
    - Ex: `BoxOzmapSource`, `CableOzmapSource`, `CustomerApiSource`, `ProjectOzmapSource`

- Localizados em: `infrastructure/datasources/[entidade]`

b. Repositories

Encapsulam a lógica de acesso a dados, orquestrando múltiplos datasources.

- **Repositórios padrão**:
    - `BoxRepository`, `CableRepository`, `CustomerRepository`, `DropCableRepository`

- **Repositórios especializados (Ozmap)**:
    - `OzmapBoxRepository`, `OzmapCableRepository`, `OzmapProjectRepository`, `OzmapProspectRepository`
      (Local: `infrastructure/repositories`)

c. Database

Gerencia a conexão com o banco de dados MongoDB.

- Arquivo principal: `MongoDbManager.ts`
  (Local: `infrastructure/database`)

d. Redis

Gerencia a fila de sincronização entre serviços e oferece mecanismos de **retry e recovery**.

- Arquivo: `RedisManager.ts`
  (Local: `infrastructure/redis`)

e. Services

Camada que executa a lógica de sincronização com fontes externas.

- `IspSyncService.ts`: integra com o sistema do ISP
- `OzmapSyncService.ts`: integra com a API Ozmap
  (Local: `infrastructure/services`)

---

4. Shared

Camada de utilitários e configurações globais.

- `config/`: configurações do ambiente (`env.ts`)
- `constants/`: constantes globais reutilizáveis
- `utils/`: funções utilitárias (ex: `logger.ts`)
  (Local: `shared/`)

---

5. Mocks

Contém simulações e dados mockados para testes e desenvolvimento local.

- Ex: `ozmap-sdk.ts`, `isp.json`
  (Local: `mocks/`)

Fluxo de Execução

1. O sistema é iniciado pelo arquivo `index.ts`.
2. Este arquivo executa o `RunServicesUseCase`, que atua como coordenador da aplicação.
3. O `RunServicesUseCase` aciona os serviços de sincronização: `IspSyncService` e `OzmapSyncService`.
4. Esses serviços utilizam repositórios específicos para consultar e salvar dados por meio dos datasources.

5. Redis entra como mecanismo de coordenação, possibilitando paralelismo, controle de tarefas, retry e resiliência.

6. Os dados atualizados são persistidos no MongoDB.

Conclusão

Meu principal objetivo para este projeto foi construir uma aplicação **escalável, testável e de fácil manutenção**. A escolha da arquitetura foi pensada para:

- Garantir **baixo acoplamento** e **alta coesão**
- Permitir **testes unitários e de integração com facilidade**
- **Isolar regras de negócio da infraestrutura**
- Facilitar a **extensão futura** com novas entidades, fontes de dados ou serviços externos

A estrutura atual está preparada para evoluir conforme as necessidades do produto, mantendo a clareza e a organização mesmo com o crescimento da base de código.