Documentação de Arquitetura por Camadas

1. Visão Geral da Arquitetura

Esta solução é composta por dois projetos principais:

1. State CSV Batch Processor
   Caminho: c:/Users/Wictorsama/source/desafio/state-csv-batch-processor

2. State Aggregator Service
   Caminho: c:/Users/Wictorsama/source/desafio/state-aggregator-service

Cada projeto segue uma arquitetura em camadas, respeitando boas práticas de manutenibilidade, escalabilidade e separação de responsabilidades.

2. State CSV Batch Processor

Propósito:
- Lê um arquivo CSV com registros de telefone.
- Agrupa e envia os dados populacionais por estado via HTTP para o serviço agregador.

Camadas da Arquitetura:

a. Camada de Entrada (Entry Point)
- Arquivo: src/main.ts
- Funções: inicia o app, configura Swagger e inicia o processamento do CSV.

b. Camada de Aplicação
- Arquivo: src/application/services/csv-reader.service.ts
- Funções: faz stream do CSV, expõe método para processar registros de forma assíncrona e trata erros.

c. Camada de Integração
- Arquivo: src/main.ts
- Funções: envia os dados em lotes para o serviço agregador via HTTP (axios).

d. Lógica de Agrupamento e Processamento
- Arquivo: src/main.ts
- Funções: valida registros, agrupa por tamanho (BATCH_SIZE) e registra logs.

3. State Aggregator Service

Documentação de Arquitetura por Camadas

Propósito:
- Recebe, agrega e armazena dados populacionais por estado.
- Expõe endpoints REST para consulta e gerenciamento.

Camadas da Arquitetura:

a. Camada de Apresentação (Controllers)
- Arquivo: src/presentation/controllers/state.controller.ts
- Funções: define os endpoints como /states/batch e /states/aggregates.

b. Camada de Aplicação (Services)
- Arquivo: src/application/services/batch-processor.service.ts
- Funções: lógica de negócio de agregação, coordenação entre controller e repositórios.

c. Camada de Domínio (Entidades e Interfaces)
- Arquivos:
  - src/domain/entities/state.entity.ts
  - src/domain/repositories/state.repository.interface.ts
- Funções: define modelos de negócio e abstrações de acesso a dados.

d. Camada de Infraestrutura (Banco de Dados)
- Arquivos:
  - src/infrastructure/database/state.repository.ts
  - src/infrastructure/database/database.module.ts
- Funções: implementações concretas para acesso ao MongoDB e definição de schemas.

e. DTOs e Validação
- Arquivo: src/application/dto/state.dto.ts
- Funções: define objetos de transferência e validação dos dados.

f. Camada de Fila (Bull + Redis)
- Arquivo: src/app.module.ts
- Funções: configura Bull com Redis para processar jobs em segundo plano, suportando retries, agendamento e escalabilidade horizontal.

4. Fluxo de Dados

1. Processamento CSV (Batch Processor)
   - Lê e valida cada linha do CSV.
   - Agrupa e envia os dados para o serviço agregador.

Documentação de Arquitetura por Camadas

2. Recepção de Lote (Aggregator Service)
   - Recebe os dados via /states/batch.
   - Enfileira os dados no Redis via Bull.
   - Agrega por estado e persiste no banco de dados.

3. Consulta dos Dados Agregados
   - Endpoints como /states/aggregates permitem recuperar dados agregados.

5. Exemplo de Interação entre Camadas

- Usuário dispara o processamento do CSV.
- Entry Point (main.ts) inicia o fluxo.
- Application Layer (CsvReaderService) processa os dados.
- Integration Layer (axios) envia os lotes via HTTP.
- Controller recebe os dados.
- Service processa e enfileira com Bull.
- Bull + Redis gerencia jobs assíncronos.
- Repository salva os dados.

6. Benefícios da Arquitetura

- Separação de Responsabilidades: cada camada tem uma função clara.
- Testabilidade: camadas desacopladas facilitam testes unitários.
- Escalabilidade: novas funcionalidades podem ser adicionadas com baixo impacto.
- Manutenibilidade: código modular, fácil de entender e manter.
- Processamento Assíncrono e Confiável: Redis + Bull garante que cargas pesadas não bloqueiem a API e sejam processadas eficientemente com suporte a retries e distribuição entre workers.