# [EPD – Engineering Planning Document] Desenvolvimento de aplicação para tratamento de dados

**Autores:**

* Anderson Bolivar Nascimento
* Anselmo Berriel de Lira
* Sanderson Esteves Vieira
* Bruno Silva Devesa
* Douglas Gonçalves Guglielmelli

**Links de Referência:**

* **DMOT Design Pattern for ETL (Data Model, Orchestrator, Transformer): https://towardsdatascience.com/dmot-a-design-pattern-for-etl-data-model-orchestrator-transformer-c0d7baacb8c7**

Links relacionados a este documento

# Linguagem Python

[O que é Python?](https://www.treinaweb.com.br/blog/o-que-e-python)

[Quais as diferenças entre tipagens](https://www.treinaweb.com.br/blog/quais-as-diferencas-entre-tipagens-estatica-ou-dinamica-e-forte-ou-fraca/)

O Python é uma linguagem de programação de alto nível, interpretada e multiparadigma, ou seja, suporta diversos paradigmas de desenvolvimento. Além do mais é uma linguagem de tipagem forte, não realiza conversões automaticamente, e dinâmica, o tipo dos dados é definido de acordo com o valor atribuído à variável em tempo de execução dinamicamente. O Python é utilizado mais frequentemente para aplicações que fazem diversas análises de dados.

[Benefícios](https://www.treinaweb.com.br/blog/quais-as-diferencas-entre-tipagens-estatica-ou-dinamica-e-forte-ou-fraca/) de se utilizar Python para a análise de dados

O Python, além de orientado a objetos (o que permite utilizar a modelagem baseada em negócios, possui uma ampla gama de bibliotecas e ferramentas para modelagem de dados, análise preditiva e estatística, desenvolvidas pela comunidade. Aliado a isto, o fato de ser uma linguagem interpretada possibilita desenvolver e corrigir erros enquanto se executa o código, diferente de Java e C++, onde o código deve ser compilado antes de ser executado e testado.

# Banco de Dados MySQL

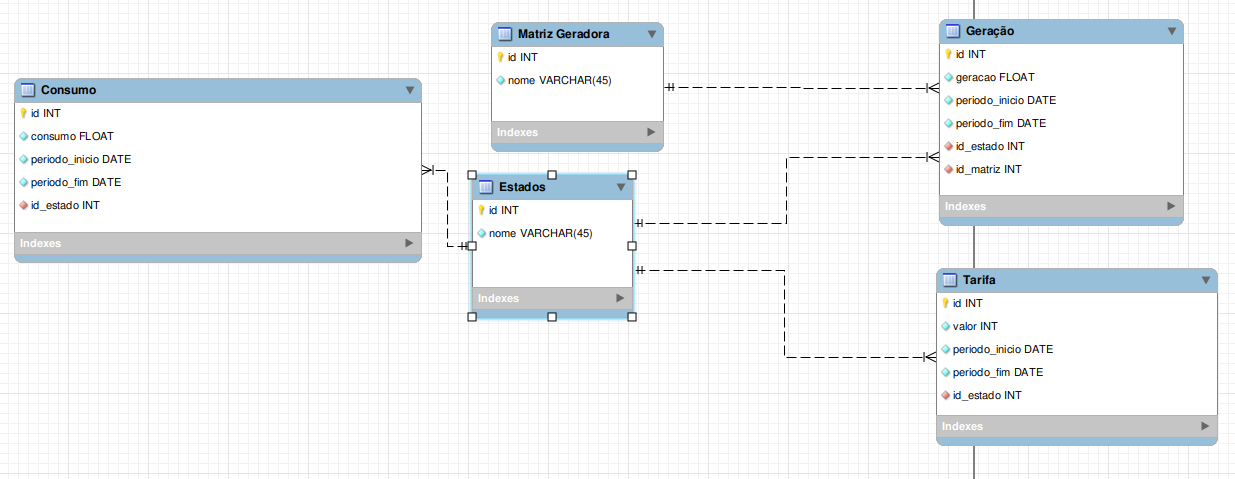
[Primeiros Passos no MySQL](https://www.devmedia.com.br/primeiros-passos-no-mysql/28438)

O MySQL é um sistema de gerenciamento de banco de dados (SGBD), que utiliza a linguagem SQL (Structured Query Language ou Linguagem de Consulta Estruturada) como interface.

**Modelo de Dados:**

Diagrama

Descrição gerada automaticamente



Arquitetura

PROPOSTA A: MVC

O MVC é um padrão de arquitetura de software utilizado em muitos projetos por possibilitar a divisão do projeto em camadas muito bem definidas. Cada uma delas, o **Model**, o **Controller** e a **View**, devem respeitar o princípio de modularidade, desacoplamento e encapsulamento entre camadas.

**Model:**

[**Connecting to MySQL Using Connector/Python**](https://dev.mysql.com/doc/connector-python/en/connector-python-example-connecting.html)

Camada responsável por gerenciar e controlar a forma como os dados se comportam por meio das funções, lógica e regras de negócios estabelecidas. Nessa cama comunicaremos com o banco e entregaremos os dados a partir de funções que farão consultas diretas no banco.

* 📁 model
  + 📁 database
    - 📄 connection.py
    - 📄 consumption.py
    - 📄 rate.py
    - 📄 generation.py
    - 📄 states.py
    - 📄 matriz.py
  + 📁 src
    - database\_a.py
    - database\_b.py

No diretório database, lidaremos com a montagem e consulta da nossa base de dados. O arquivo connection.py fará a instância do MySQL no projeto, utilizando a lib mysql. O restante dos arquivos possuirá funções que se comunicarão com o banco.

No diretório src, faremos a extração e limpeza das bases de dados externas escolhidas, utilizando a lib requests, teremos um arquivo para cada base de dados.

**Controller:**

O controller é a camada responsável por intermediar as requisições enviadas pelo View com as repostas fornecidas pelo Model. Nele isolaremos toda nossa regra de negócio, ou seja, as análises que faremos de transformação de dados.

* 📁 controller
  + 📄 index.py
  + 📄 reports.py
  + 📄 update.py

O arquivo index.py vai apenas exportar os outros dois arquivos da camada. O reports.py fica concentrado toda a lógica de análise de dados que faremos, o relacionamento entre as entidades. É interessante que essas funções tenham testes e comentários descrevendo os seus parâmetros e resultado. O update.py será o arquivo que vai popular o banco de dados com os dados brutos extraídos e limpados pela camada model, esse arquivo deve ser interpretado apenas uma vez durante o ciclo de vida da aplicação.

**View:**

Camada que irá apresentar as informações de forma visual ao usuário.

Existem várias bibliotecas do Python de visualização de dados, devemos escolher qual atende melhor os nossos critérios:

[**Python Data Visualization Libraries for Business Analytics**](https://mode.com/blog/python-data-visualization-libraries/)

Caso necessário, podemos usar também o *inquirer* para interação com o usuário via CLI.

* 📁 view

# PROPOSTA B: Proposição de arquitetura para o projeto

[Common Big Data Design Patterns](https://hub.packtpub.com/common-big-data-design-patterns/)

[Big Data Architectural Patterns](https://www.slideshare.net/AmazonWebServices/big-data-architectural-patterns-and-best-practices)

[10 Common Software Architectural Patterns in a nutshell (O PRINCIPAL)](https://towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013)

Para o desenvolvimento da solução de tratamento e análise dos dados, propomos seguir a arquitetura de soluções ETL (Extract, Transform and Load) sob o padrão [Pipe-filter pattern](https://towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013).

O padrão de processo ETL (de Extract, Transform and Loading) segue um design já conhecido no mercado de processos para carga, tratamento e exibição de dados, conforme exemplificado na imagem abaixo:

Interface gráfica do usuário

Descrição gerada automaticamente

O processo de ETL especifica ao menos 3 etapas no processo de tratamento de dados, cada uma realizada em uma parte do projeto em si:

1. Ingest: etapa realizada por scripts (no nosso caso, desenvolvidos em Python), que buscam ler os arquivos de dados e processá-los, unificando em uma única base de dados. Assim, irá gerar entradas em tabelas de banco de dados com o fim de transformar as diversas informações distintas em um contexto único (isto é, as diferentes tabelas terão que dar uma única informação, que é o consumo de energia para uma dada localidade em dado período de tempo e por matriz geradora).
2. Transformação (tratamento): nesta etapa, os dados unificados são limpos, tratados e processados de forma a facilitar buscar as informações desejadas (no nosso caso, variações de tarifa com verificação de mudanças de uso de matrizes geradoras).
3. Análise e exibição: nesta etapa, os dados serão submetidos a análises por meio de consultas e exibidos, utilizando algum programa para montagem de gráficos e dashboards (como o PowerBI ou o Tableau).

O padrão de projeto Pipe-Filter Pattern é um padrão já conhecido para execução de processos em fila, onde a execução é iniciada assim que algo acontece (ou alguém manda iniciar) e conclui após processar a solicitação ou terminar de processar todos os dados envolvidos.

[Diagrama

Descrição gerada automaticamente](https://static.packt-cdn.com/products/9781789809770/graphics/assets/ab9be4ee-22f4-4f04-982f-3b04a754468c.png)

As etapas do projeto correspondente a parte de aplicação (onde estarão os scripts python e a aplicação de exibição) serão desenvolvidos sob o padrão Layered Pattern, do qual serão utilizados somente 4 camadas: Presentation Layer, Business Layer, Persistence Layer e Database Layer.

[Diagrama

Descrição gerada automaticamente](https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/assets/sapr_0103.png)

1. Business Layer

A camada de negócios (ou business layer) contêm os códigos que realizam as ações relacionadas ao negócio. Todas as regras da aplicação e a parte de código responsável por elas se encontra nesta parte.

No nosso caso, os scripts Python responsáveis pela manipulação dos dados lidos dos arquivos Excel e do banco de dados correspondem à Business Layer, pois os mesmos serão responsáveis por explorar e analisar os dados em busca das informações que queremos obter.

[Uma imagem contendo Diagrama

Descrição gerada automaticamente](https://smist08.files.wordpress.com/2010/07/arch.png)

1. Persistence Layer

A camada de persistência de dados (ou persistence layer) conterá a parte do código que realiza as operações de criação de dados, atualização de dados, consulta e exclusões (também conhecido como CRUD). Esta camada é responsável exclusivamente por falar com o banco de dados (no nosso caso, o MySQL).

No exemplo da imagem abaixo, os 2 pedaços de código de nomes BuyerRepository e OrderRepository utilizam um conceito chamado [Unit of Work](https://www.c-sharpcorner.com/UploadFile/b1df45/unit-of-work-in-repository-pattern/) para realizar as operações de inserir, consultar, excluir e atualizar no banco.

Diagrama

Descrição gerada automaticamente

1. Database Layer

Nesta camada, temos o banco de dados em si, no nosso caso o MySQL com as tabelas conforme especificado mais acima. O banco será acessado tanto pelos scripts de tratamento de dados quanto pela aplicação de visualização, o Tableau, na camada de apresentação conforme especificado abaixo.

1. Presentation Layer

A camada de apresentação, conforme apresentada na figura abaixo, é onde o usuário tem quaisquer visualizações do que acontece. No nosso caso, a única porção de apresentação que teremos na nossa solução será o Tableau, através do qual os dados já analisados serão explorados e as visualizações em gráficos serão montadas.

[Uma imagem contendo Interface gráfica do usuário

Descrição gerada automaticamente](https://i.stack.imgur.com/gHL5h.png)

Conforme especificado acima, o tableau será nossa camada de visualização em um fluxo de análise de dados. Assim, o esquema abaixo representa o que mais se aproxima do que faremos:

[Diagrama

Descrição gerada automaticamente](https://miro.medium.com/max/1400/1*s-JZVy7xMYUGA83SFAht3Q.jpeg)

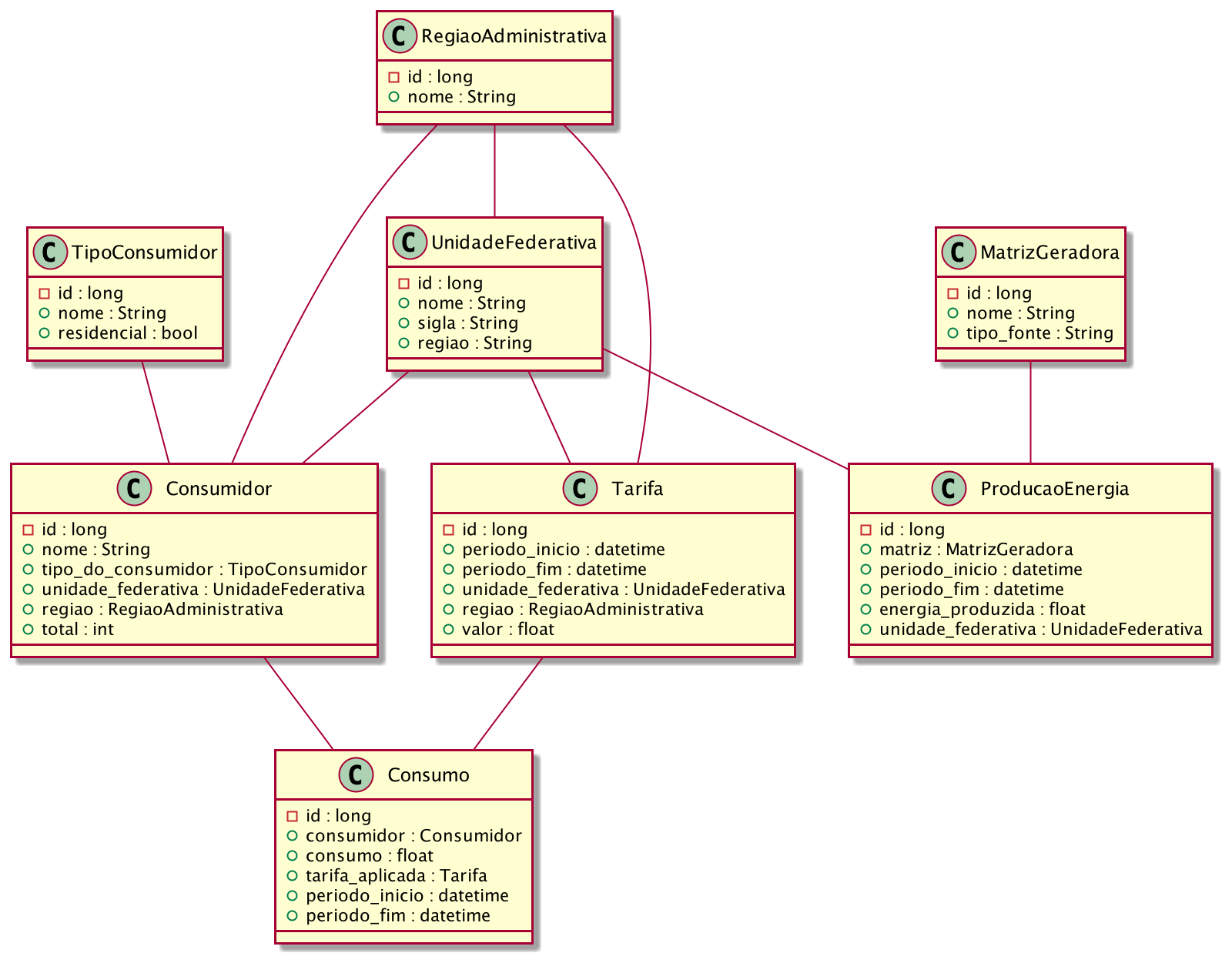
Como pode ser observado na imagem acima temos:

1. Parte esquerda: as fontes de dados (no nosso caso, são arquivos Excel obtidos de diferentes sites de dados sobre energia elétrica). Assim, o item “File System” é o que atende nosso negócio.
2. Parte central: esta parte central se divide em 3 partes:
   1. Source DB: base de dados onde os dados extraídos dos arquivos excel serão guardados. Antes disso, iremos limpar as sujeitas e agrupar as diferentes informações nos diferentes arquivos em uma única: qual o consumo de energia de determinado tipo de consumidor em uma dada data para o local dele e qual a tarifa aplicada ali naquele instante de tempo.
   2. Business Logic Transformation: onde os dados serão analisados e extrairemos o máximo de informações o possível para poder, depois, calcular as variações de tarifas e determinar onde está ocorrendo aumento de fato e o motivo.
   3. Target DB: a base de dados onde os dados já analisados serão guardados. No nosso caso, em princípio, temos um schema de banco único onde os dados serão guardados após o ingest e também após a análise.
3. Parte direita: visualização dos dados com Tableau.

# Especificações de modelos e arquitetura em diagramas

Modelo de Classes

Abaixo, segue o modelo de classes da aplicação, o qual dará origem ao banco de dados com os dados agrupados e também tratados.



Segue o dicionário de dados:

* RegiaoAdministrativa
  + Definição: aqui temos o cadastro das regiões administrativas do Brasil (Norte, Nordeste, Sudeste, Centro-Oeste e Sul). Como alguns dados de consumo estão catalogados por região, esta classe/tabela receberá esses dados logo após o tratamento.

|  |  |  |
| --- | --- | --- |
| Coluna | Tipo | Definição |
| id | long | Identificador único do registro no sistema |
| nome | String | Nome da região administrativa |

* UnidadeFederativa
  + Definição: registro das unidades federativas (ou estados) do Brasil.

|  |  |  |
| --- | --- | --- |
| Coluna | Tipo | Definição |
| id | long | Identificador único do registro no sistema |
| nome | String | Nome da unidade federativa |
| sigla | String | Sigla da unidade federativa (para exibição) |
| regiao | String | Região à qual a unidade federativa pertence |

* TipoConsumidor
  + Definição: aqui temos o registro das categorias de consumidores por tipo de usuário (pessoa física - residencial – ou pessoa jurídica - setores produtivos ou comércio).

|  |  |  |
| --- | --- | --- |
| Coluna | Tipo | Definição |
| id | long | Identificador único do registro no sistema |
| nome | String | Nome do tipo ou categoria de consumidor |
| residencial | Boolean | Se o consumidor é residencial ou não |

* Consumidor
  + Definição: aqui temos o registro dos tipos de consumidores por região e a quantidade presente. Esta segregação por tipo permite registrar o histórico por cada tipo e também para todos.

|  |  |  |
| --- | --- | --- |
| Coluna | Tipo | Definição |
| id | long | Identificador único do registro no sistema |
| nome | String | Nome do consumidor (ex: “Baixa Renda” ou “Comércio de Rua”) |
| tipo\_do\_consumidor | TipoConsumidor | Tipo ou categoria do consumidor (se é pessoa física, industrial, etc) |
| unidade\_federativa | UnidadeFederativa | Estado ou unidade da federação à qual o consumidor pertence |
| regiao | RegiaoAdministrativa | Para os casos em que a informação da unidade da federação não existe mas sim a da região, este campo será preenchido. |
| total | inteiro | Contagem de consumidores deste tipo |

* Tarifa
  + Definição: aqui temos o histórico de tarifas aplicadas sobre energia consumida por período (dias, horas, meses, etc) e região. Como os indicadores de tarifas ora vem por unidade de federação, ora vem por região do país, os 2 foram inclusos no modelo (mas um deles estará preenchido necessariamente).

|  |  |  |
| --- | --- | --- |
| Coluna | Tipo | Definição |
| id | long | Identificador único do registro no sistema |
| período\_inicio | datetime | Data inicial do período de vigência da tarifa |
| período\_fim | datetime | Data final do período de vigência da tarifa |
| unidade\_federativa | UnidadeFederativa | Unidade federativa ou estado à qual a tarifa foi aplicada (para o caso de informação de tarifa encontrada por estado da federação) |
| regiao | RegiaoAdministrativa | Região do país (norte, nordeste, sudeste, centro-oeste, sul) à qual a tarifa foi aplicada (para o caso de informação da tarifa encontrada por região) |
| valor | float | Valor da tarifa aplicada |

OBS: como atividade da análise de dados, as tarifas ao final estarão aplicadas aos estados (mesmo que tenha vindo por região), mas para o caso de tarifa obtida por região, o campo de região estará devidamente preenchido.

* Consumo
  + Definição: aqui temos o histórico de consumo por tipo de consumidor, região e a tarifa aplicada na data/período do consumo registrado e região do consumidor.

|  |  |  |
| --- | --- | --- |
| Coluna | Tipo | Definição |
| id | long | Identificador único do registro no sistema |
| consumidor | Consumidor | Nome da região administrativa |
| consumo | float | Total de energia consumido |
| tarifa\_aplicada | Tarifa | A tarifa aplicada a este consumo |
| período\_inicio | datetime | Data inicial do período analisado para gerar o consumo registrado |
| período\_fim | datetime | Data final do período analisado para gerar o consumo final |

* MatrizGeradora
  + Definição: aqui temos o registro das matrizes geradoras de energia do país e a definição do tipo da mesma.

|  |  |  |
| --- | --- | --- |
| Coluna | Tipo | Definição |
| id | long | Identificador único do registro no sistema |
| nome | String | Nome da matriz geradora (empresa, etc) |
| tipo\_fonte | String | Nome do tipo de fonte energética utilizado (se elétrica, combustível fóssil, solar, etc) |

OBS: As matrizes geradoras são as empresas ou órgãos responsáveis pela produção de energia e eles poderão atuar em mais de um estado com instalações distintas. Por isto que o registro de matrizes geradoras não possui o indicativo da região do país ou unidade da federação.

* ProducaoEnergia
  + Definição: aqui temos o registro histórico de produção energética por matriz e região (pode ser unidade federativa ou região administrativa, pois nas fontes de dados podemos encontrar nos 2 formatos).

|  |  |  |
| --- | --- | --- |
| Coluna | Tipo | Definição |
| id | long | Identificador único do registro no sistema |
| matriz | MatrizGeradora | Matriz geradora responsável pela produção de energia |
| período\_inicio | datetime | Data inicial da contagem de energia produzida |
| período\_fim | datetime | Data final da contagem de energia produzida |
| energia\_produzida | float | Quantidade de energia produzida (em MW) |
| unidade\_federativa | UnidadeFederativa | Estado da federação em que a energia foi produzida. |

OBS: para matrizes geradoras presentes em diversos estados, é bom colocar qual a unidade da federação em que a energia foi produzida.

Diagrama de Atividades

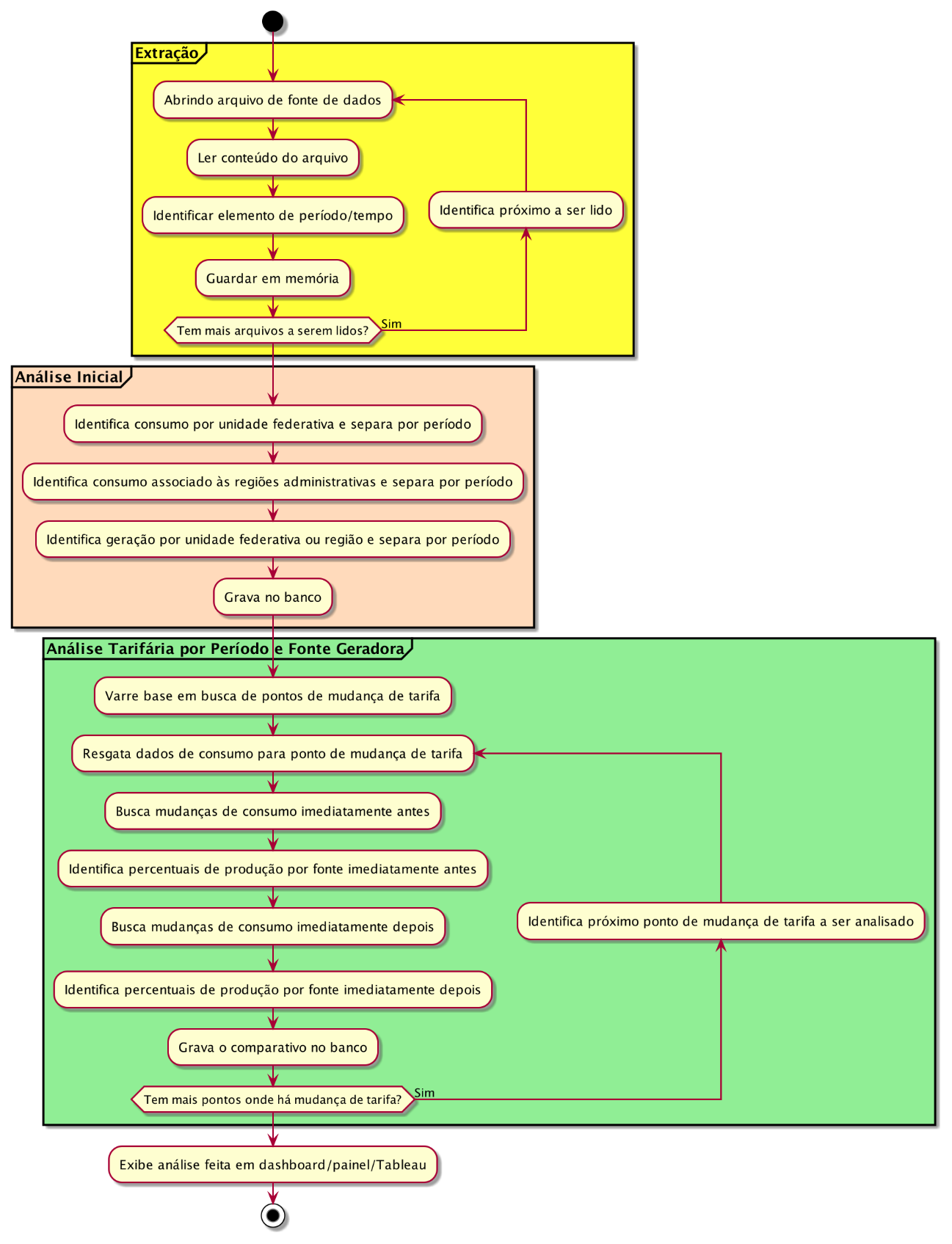
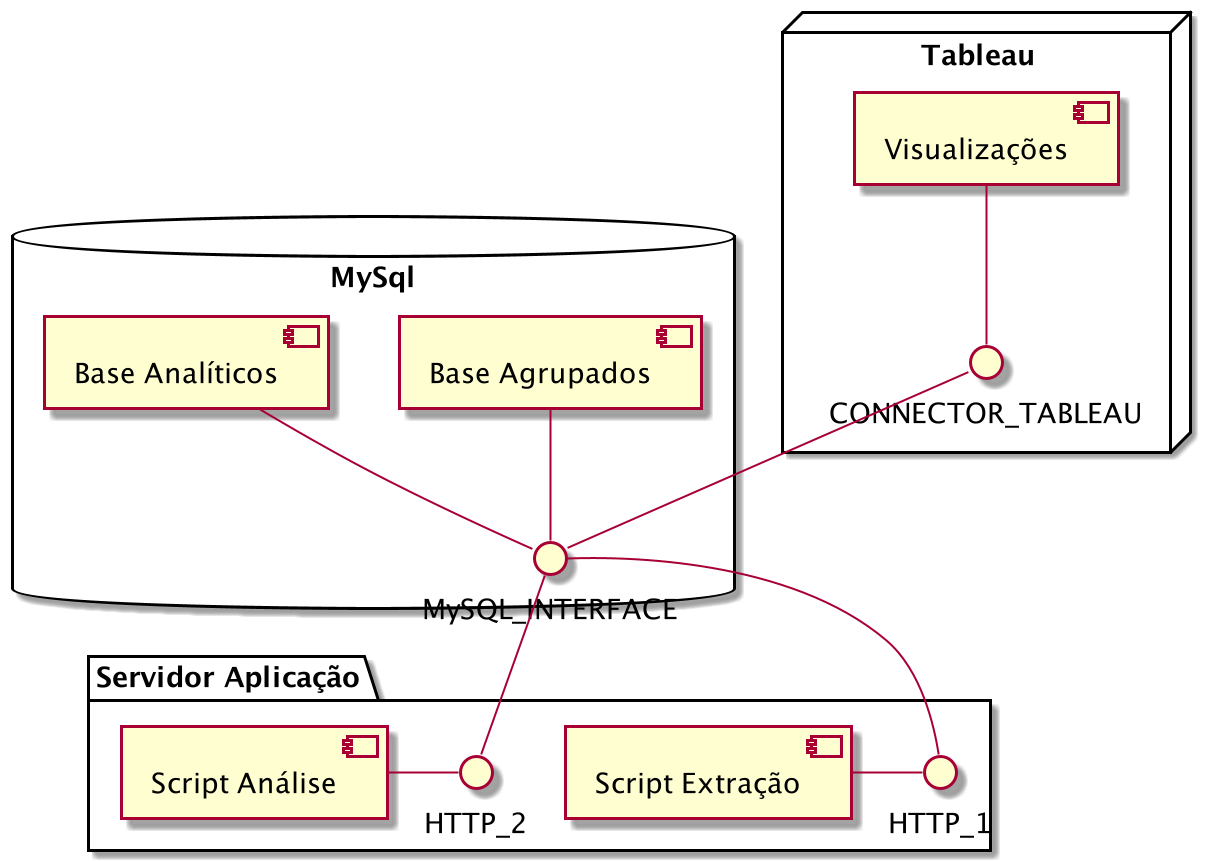


Diagrama de Infraestrutura



Analisando o esquema acima, podemos ver claramente as 4 camadas do padrão Layered Pattern:

1. Presentation Layer: Tableau
2. Business Layer: “Script Análise” e “Script Extração”
3. Persistence Layer: incluso em “Script Análise” e “Script Extração”
4. Database Layer: presente em “MySQL”