Arquitetura e desenvolvimento de um produto de dados com informações de candidatos a cargos públicos: "iDEMOCRACIA"

João Paulo Silva Simões[1]

RESUMO

Considerando que a disponibilização de dados de públicos é feita de forma bruta em planilhas eletrônicas, e a obtenção de informações tratadas é comprometida por este formato, sugere-se uma aplicação para filtragem e apresentação dessas informações de maneira estrutura, sendo assim este artigo apresenta o desenvolvimento de um produto de dados intitulado "iDEMOCRACIA" para consulta de dados públicos, de maneira unificada e relacionada, a partir dos dados fornecidos pelo Estado Brasileiro, buscando estreitar a relação entre Estado e Cidadão, quer seja por meio do indivíduo, entidades de classe, jornalistas ou acadêmicos. Tal aplicação visa simplificar o acesso a informações para o acompanhamento, análise, investigação, tomada de decisões, ou o desenvolvimento de subprodutos úteis a sociedade. Utilizou-se de tecnologias e conceitos tais como Big Data, sendo elaborado através de uma pesquisa e análise exploratória, avaliando os dados e informações disponibilizadas, sob a ótica da ciência de dados, e o resultado é uma plataforma para a democratização e consulta de dados.

Palavras-chave: Ciência de dados, Python, Amazon Web Services, Eleições, Produto de Dados.

1 INTRODUÇÃO

Atualmente, nos encontramos na era da informação, período pós-industrial, a partir da década de 70, na qual nossa sociedade está interligada e imersa em dados e informações, em muito devido a popularização dos microprocessadores, avanços em telecomunicações, e a miniaturização de componentes, possibilitando o acesso rápido e barato as mais diversas fontes e formas de informação em escala global, virtualmente a todos os habitantes do planeta de maneira ilimitada e ininterrupta (ROCHA, BASTOS, CARVALHO, 2020).

Entretanto, tal facilidade de acesso aos dados e informações, por uma parcela significativa da população, não resultou de maneira equivalente em uma maior participação da mesma frente a gestão pública e a escolha de seus representantes. Embora tais informações encontrem-se em domínio público, são por muitas vezes descentralizadas e pouco claras, como exemplo podemos citar o PBDA[2] e suas diversas fontes de dados não relacionadas.

Com o intuito de facilitar e aproximar a sociedade civil da gestão pública, por meio da unificação e relacionamento de seus dados, objetiva-se a construção de uma plataforma livre de fácil acesso e compreensão, por parte do cidadão comum por meio de portais ou aplicativos moveis, quer seja por jornalistas, acadêmicos ou pesquisadores através do acesso público a uma API (*Application Programming Interface*) ou Interface de Programação de Aplicativos, baseada no padrão REST (*Representational State Transfer*).

---

[1] Gerente de Projetos, Bacharel em Sistemas de Informação, pós-graduado em Gerenciamento de Projetos pela Pontifícia Universidade Católica de Minas Gerais (PUC-MG). E-mail: joao.simoes@sga.pucminas.br

[2] Portal Brasileiro de Dados Abertos, disponível em: https://dados.gov.br

Sendo assim, optou-se pela abordagem *API-First Design Pattern* (VESTER, 2022), ou em tradução livre, padrão de projeto primeiramente API, no qual o foco é o desenvolvimento de produtos ou soluções a orbita de uma APIs desde sua concepção, possibilitando assim uma visão mais clara dos desafios a serem superados e objetivos a serem alcançados.

Desta forma, o projeto utilizará somente uma fração das fontes de dados públicas disponíveis, para que sejam avaliadas as características e desafios técnicos, bem como a viabilidade e prototipação de sua arquitetura. Para tal, será utilizada uma fonte de dados estável e consolidada, o repositório de dados eleitorais relativo ao pleito de 2016, representando por si só, um grande desafio, uma vez que estes encontram-se em formatos e fontes de dados heterogêneas.

Neste sentido, o objetivo central do projeto, é o desenvolvimento de um produto de dados e sua arquitetura, para o acesso universal as informações relativas ao Estado Brasileiro, quer seja em seu âmbito federal, estadual ou municipal, a partir de um produto de dados, disseminando o acesso à informação.

Este artigo está estruturado da seguinte forma: a Seção 2 trata do desenvolvimento do estudo, as tecnologias utilizadas e como estão integradas a fim de se alcançar os objetivos deste projeto, bem como a modelagem das informações e refino dos dados brutos, e por fim, na Seção 3 são relatadas as considerações finais.

2 DESENVOLVIMENTO

Para o desenvolvimento da aplicação será necessária uma arquitetura igualmente robusta, que possibilite a coleta, tratamento, armazenamento e distribuição de toda essa cadeia de informações, sendo assim serão descritas as tecnologias aplicadas e os desafios enfrentados no âmbito do projeto.

Os dados por sua vez são disponibilizados em sua forma bruta como arquivos compactados e/ou arquivos de texto, sendo necessário além de sua obtenção e extração, sua posterior transformação, para que seja possível realizar sua inserção junto a uma base de dados relacional devidamente modelada, que será efetivamente utilizada pela API REST. Processo este conhecido como ETL (*Extract, Transform and Load*), ou Extração, Transformação e Carga, utilizando uma linguagem de desenvolvimento de alto nível.

No que tange o armazenamento dos dados durante a elaboração da arquitetura da solução, optou-se por uma abordagem hibrida, onde uma vez os dados obtidos junto a fonte primária, estes ainda em sua forma bruta seriam preservados junto a um *Data Lake* ou lago de dados, sendo posteriormente utilizados como o principal insumo para os processos de ETL, e posterior composição de uma base de dados relacional.

Uma vez que se pretende mais do que disponibilizar o mero acesso aos dados em um novo formato, busca-se descobrir e evidenciar as associações entre os entes envolvidos, acompanhar a evolução patrimonial de indivíduos, traçar correlações entre micro e macrorregiões, além de uma análise detalhada e categorizada dos investimentos em campanha, bem como demais inferências e conjecturas possíveis acerca do tema.

2.1 Ferramentas de ETL

A obtenção dos dados brutos por si, não caracterizam o maior desafio do projeto, tendo em vista que tais dados são de domínio públicos e possuem fácil acesso por meio de ferramentas e soluções dedicadas aos processos de ETL, quer sejam estas comerciais ou *open source*, também conhecidas como programas de código aberto.

Entretanto, muitas vezes, tais soluções ao mesmo tempo em que possuem grande apelo visual e contam com diversos recursos gráficos que facilitam sua utilização por parte dos usuários podem também limitar o escopo da implementação de soluções de ETL robustas e distribuídas.

Muitas vezes não possuindo integração externa com outras soluções por meio de CLI (*Command Line Interface*), ou Interface de Linha de Comando, quando não, são desenvolvidas para funcionar exclusivamente junto a sistemas operacionais predeterminados como, por exemplo, as soluções SSIS (*SQL Server Integration Services*) e *Power BI* da *Microsoft* ou *Tableu* da *Tableu Software*, embora muitas destas possuam foco na visualização e análise de dados, por vezes também são utilizadas como ferramentas de ETL.

Sendo assim, optou-se pelo desenvolvimento de um ETL próprio, se valendo da linguagem de desenvolvimento de código aberto *Python*, em sua versão 3.8, devido à sua aderência e características junto ao escopo deste projeto, dentre estas se destacam: simplicidade, boa expressividade, vasta documentação, multiplataforma, grande número de bibliotecas de propósito geral e dedicadas a análise e tratamento de grandes volumes de dados.

Sendo adotada pela comunidade de *Big Data* e *Data Science* (DATACAMP, 2022), possuindo diversas bibliotecas e serviços que a suportam, dentre as utilizadas no projeto destacamos *Pandas* e *SQLAlchemy*.

Porém assim como uma matéria bruta que possui grande potencial, a depender somente do esforço empenhado em seu beneficiamento, assim é com o uso de uma linguagem de programação para o desenvolvimento de um processo de ETL, pois implica em um maior esforço, já que todas as etapas devem ser implementadas manualmente, reduzindo consideravelmente os ganhos obtidos quando se utilizam ferramentas gráficas que abstraem tais etapas deste processo.

Por outro lado, são colhidos os benefícios de tal abordagem, uma vez que podem ser executadas operações extremamente refinadas sobre os dados coletados, até mesmo durante o processo de carga e ajustes quanto o volume dos dados a serem processados, podendo inclusive ser delegado a sistemas distribuídos sem grandes esforços.

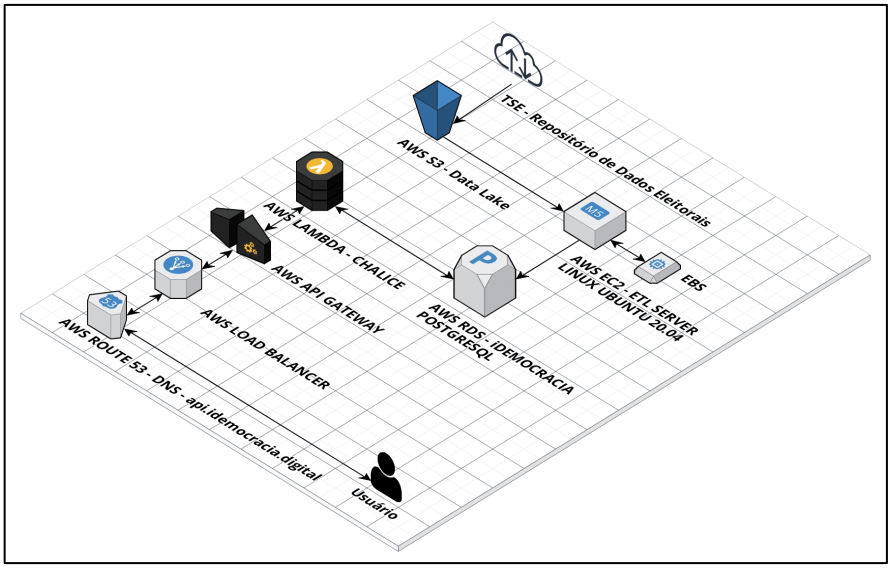
2.2 Infraestrutura de nuvem

A infraestrutura é um ponto crítico para o projeto, sendo assim, optou-se pela utilização dos serviços de PaaS (*Plataform as a Service*) ou Plataforma como Serviço, por meio do provedor *Amazon,* e sua suíte dedicada a este fim, denominada AWS (*Amazon Web Services*) (AWS, 2022).

A escolha deste provedor se deu devido a sua ampla gama de serviços, vasta documentação, e cobertura mundial, contando com *data centers* em território nacional, potencializando assim seu alcance caso o projeto prospere.

Porém para uma melhor compreensão apresenta-se na Figura 1 a estrutura, bem como os serviços e componentes que serão utilizados no projeto, e seu papel junto à arquitetura de nuvem.

Figura 1 – Diagrama de Arquitetura de Nuvem – AWS



Fonte: Elaborada pelo autor

No Quadro 1, há o detalhamento (descrição e aplicação) das tecnologias utilizadas na implementação na arquitetura AWS.

Quadro 1 – Tecnologias de Arquitetura de Nuvem – AWS

Segmento Tecnologia	**Descrição**	**Aplicação**
Armazenamento AWS S3	O *AWS S3* (*Simple Storage Service*) é o serviço de armazenamento de arquivos provido pela AWS, capaz de armazenar grandes volumes de dados em uma estrutura simplificada e distribuída, organizada em *buckets* ou baldes que são análogos a pastas (AWS S3, 2022).	Tal serviço possibilitará armazenar os dados brutos coletados junto a nossas fontes de dados, que atuará como *Data Lake* em nosso projeto, em uma etapa intermediaria, também conhecida como *Staging Area*, conceito este proveniente da arquitetura de DW (*Data Warehouse*) e BI (*Business Inteligence*).
Rede AWS Route 53	O AWS Route 53 é o serviço de DNS (*Domain Name System*) ou sistema de nome de domínios, responsável por resolver ou traduzir os nomes de endereço digitados em endereços IP (*Internet Protocol*), o serviço organiza os domínios por meio de *hosted zones* ou zonas hospedada (AWS ROUTE 53, 2022).	Tal serviço possibilitará a resolução das requisições realizadas junto ao domínio escolhido (https://api.idemocracia.digital) e seu correto encaminhamento aos serviços desejados, visando mitigar os riscos relacionados à utilização de endereços IPs, tendo em vistas que estes endereços são voláteis e quando modificados podem acarretar na quebra de compatibilidade das chamadas à API, resultando em um grande esforço para refatoração de códigos fontes preexistentes, atualização de documentação técnica, bem como uma série de outros riscos não elencados, que podem ser mitigados com sua simples adoção.
Computação AWS EC2	O AWS EC2 (*Elastic Compute Cloud*) é o serviço de processamento computacional, no qual é possível realizar a configuração, iniciação, redimensionamento, escalonamento e monitoramento de servidores e sua	Tal serviço possibilitará a solução alocar a capacidade computacional necessária para realização dos processos de ETL de maneira segura e isolada, sem que sejam comprometidos os demais recursos, componentes e serviços de nossa arquitetura,

	infraestrutura, de maneira simplificada e unificada. O serviço organiza tais recursos por meio de *instances* ou instâncias, que são análogos a equipamentos físicos (AWS EC2, 2022).	uma vez que estes estarão alocados e dedicados ao armazenamento e gestão do banco de dados relacional, disponibilização da API pública, análise e visualização das informações, bem como as demais funcionalidades.
Rede AWS ALB	O AWS ALB (*Application Load Balancer*) é o serviço para centralização, recebimento e encaminhamento de requisições aos serviços fornecidos, possibilitando assim uma interface comum para acesso o externo. Garantindo ao produto um maior nível de segurança, uma vez que tal recurso promove o isolamento da infraestrutura interna, impossibilitando o acesso direto aos servidores ou seus endereços IPs, mitigando em grande parte potenciais ataques do tipo DDoS (*Distributed Denial of Service*) ou ataque distribuído de negação de serviço, implementando em um único ponto uma camada segura de comunicação, por meio do protocolo HTTPS (*Hyper Text Protocol Secure*) (AWS ALB, 2022).	Tal serviço possibilitará a aplicação ser integrada junto ao sistema de nome de domínios utilizado, no caso concreto o Route 53, onde todas as requisições realizadas ao serviço serão encaminhadas ao ALB, que terá como funções: isolar a infraestrutura interna, provendo um canal de comunicação seguro por meio do protocolo HTTPS e realizar o balanceamento de carga e alocação de recursos computacionais conforme o aumento do número de usuários e acesso a seus serviços.
Rede AWS API Gateway	O AWS API Gateway é o serviço dedicado a gestão e controle de APIs, fornecendo uma vasta gama de funcionalidades para sua criação, documentação, manutenção, publicação, cache e escalonamento, permitindo monitorar o acesso a todos os recursos disponibilizados, bem como a possibilidade de definição de cotas de uso ou revogação de privilégios (AWS API GATEWAY, 2022).	Tal serviço fornecerá acesso seguro a todos os usuários que venham a requisitar acesso aos serviços fornecidos por nossa API REST, e que estas requisições por sua vez sejam corretamente distribuídas em uma arquitetura resiliente e escalável baseada em *microservices*, ou microsserviços.
Computação AWS Lambda	O AWS Lambda é o serviço dedicado a computação em nuvem distribuído, baseado na arquitetura de *microservices*, ou microsserviços, o qual fornece escalabilidade, alta disponibilidade, desenvolvimento poliglota (suporte a diversas linguagens de desenvolvimento), e integração junto aos demais serviços da AWS (AWS LAMBDA, 2022).	A utilização do serviço possibilitará o desenvolvimento de uma API com alta disponibilidade e escalabilidade, sem a necessidade de alocação previa de recursos computacionais, que por ventura poderiam se tornar ociosos em momentos com um menor número de acessos e requisições, ao mesmo tempo que fazendo uso desta arquitetura caso o serviço sofra uma grande demanda em períodos sazonais, como as vésperas de pleitos eleitorais a solução já se encontrará apta a atender em tempo hábil a grande carga de trabalho recebida.
Desenvolvimento AWS Chalice	O AWS Chalice é um framework ou estrutura de código e conjunto de ferramentas que permitem o desenvolvimento, distribuição e publicação de aplicações baseadas em *Python*, junto ao serviço de computação distribuída AWS Lambda, simplificando o fluxo de trabalho de	Por meio de tal recurso será possível o desenvolvimento da infraestrutura necessária a API RESTful do projeto, sem que seja necessário provisionar significativos recursos computacionais, uma vez que as cobranças decorrem somente por meio da quantidade de chamadas realizadas, assim como seu escalonamento, que é

	maneira integrada aos demais serviços necessários tais como AWS SDK, AWS IAM e AWS CLOUDFRONT (AWS CHALICE, 2022).	realizado de maneira automatizada conforme mais recursos são necessários.
Banco de Dados AWS RDS	O AWS RDS (*Relational Database Service*) é o serviço dedicado a criação, configuração e gestão de todo ciclo de vida de bases de dados relacionais, pelo qual é possível simplificar sua distribuição, escalonamento e preservação, suportando diversos produtos e fornecedores (AWS RDS, 2022).	Por meio da utilização deste serviço será possível a gestão simplificada da base de dados relacional utilizada junto ao projeto, optando-se pela utilização da *engine* ou motor PostgreSQL em sua versão 12 (POSTGRESQL, 2022).

Fonte: Elaborada pelo autor

2.3 Arquitetura REST e RESTful

O padrão REST (*Representational State Transfer*) ou transferência de estado representacional, foi proposto por Roy Fielding (FIELDING, 2000), é considerado um modelo de arquitetura de *software* para o desenvolvimento de interfaces de comunicação entre sistemas, utilizando o já estabelecido protocolo HTTP (*Hypertext Transfer Protocol*), baseando-se em seu conceito original de acesso a recursos por meio de URIs (*Uniform Resource Identifiers*) e ações por meio de suas operações ou verbos, como: *GET, HEAD, POST, PUT, PATCH, DELETE, CONNECT, OPTIONS* e *TRACE* (LIMA, 2020).

Entretanto a aderência ao modelo proposto não necessariamente implica na adoção de todos os seus 6 (seis) princípios, ou restrições arquitetônicas (IBM, 2022), embora seja recomendado ao menos a adoção de 5 (cinco) destes, para que uma API possa ser considerada RESTful (RESTFUL API, 2022), sendo eles:

1. ***Uniform Interface*** (ou Interface Uniforme): Princípio no qual o acesso aos recursos deverá ser único, independentemente de onde os mesmos estiverem sendo requisitados, devendo conter em sua resposta todas as informações relevantes relativas ao recurso (IBM, 2022). Tal feito deve ser apresentado de forma parcimoniosa, ou seja, de maneira não tornar excessivo o trafego de dados, porém para contornar tal limitação recomenda-se o uso do mecanismo de hipermídia, por meio da implementação de HATEOAS (*Hypermedia as the Engine of Application State*), ou hipermídia como o mecanismo do estado do aplicativo (GUPTA, 2022);
2. ***Client-Server*** (ou Cliente-Servidor): Princípio no qual deve-se haver uma separação clara entre os dois entes de maneira tal, permitir a evolução independente de cada qual, sendo somente a comunicação estabelecida por meio de acesso aos recursos através de URIs (IBM, 2022);
3. ***Statelessness*** (ou Sem Estado): Princípio no qual as requisições aos recursos podem ser realizadas de maneira autossuficiente, ou seja, sem necessitar que o Estado esteja preservado junto ao Servidor, possibilitando que o mesmo seja escalável e facilmente distribuído (IBM, 2022);
4. ***Cacheability*** (ou Possibilidade de Cache): Princípio no qual deve-se permitir quando possível que as respostas as requisições URIs sejam passíveis de armazenamento em cache, buscando otimização do tempo de resposta e melhoria da experiência dos usuários (IBM, 2022);

5. ***Layered System Architecture*** (ou Sistema em Camadas): Princípio no qual uma API deve ser estruturado em camadas, não devendo ser possível ao cliente acessar nenhum recurso além dos disponibilizados na camada que o mesmo está interagindo (IBM, 2022);
6. ***Code on Demand*** (ou Código Sob Demanda): Princípio no qual pode ser possível o envio de código a ser executado junto ao Cliente, diferente dos recursos estáticos comumente providos por uma API, tal princípio é considerado opcional (IBM, 2022).

Quanto ao projeto foram aplicados os 5 (cinco) princípios inicialmente descritos, de maneira a prover uma API RESTful sob os princípios previamente estabelecidos.

2.4 Especificação OPEN API ou Swagger

O OPEN API, também conhecido como especificação Swagger, atualmente em sua versão 3 (três), é um padrão internacional mantido pela *OpenAPI Initiative*, instituto dedicado a disponibilizar um formato comum para documentação e descrição de APIs e seus serviços, de maneira agnóstica, ou seja, sem dependência a fornecedores ou agentes externos (OPEN API, 2022).

Utilizando de linguagem JSON (*JavaScript Object Notation*) ou YAML (Yet Another Markup Language), é possível descrever de maneira clara e objetiva, todos os recursos e serviços disponibilizados pela mesma, possibilitando a integração com diversos provedores de serviços de computação em nuvem, dentre eles a AWS e o serviço *Amazon API Gateway***.**

2.5 Mecanismo HATEOAS e HAL

O conceito do mecanismo de HATEAOS, é uma etapa essencial para se prover expressividade a uma API, ou seja, disponibilizar informações e documentação dinâmica ao cliente, o qual poderá navegar por dentre as funcionalidades diretamente fornecidas pelos recursos, sem a necessidade explicita de documentação adicional (GUPTA, 2021).

Entretanto existem diversas possibilidades para sua implementação, dentre as quais destacamos a HAL (*Hypertext Application Language*), ou linguagem de aplicação de hipertexto, que tem por objetivo fornecer uma convenção simplificada e direta para declaração de hiperlinks através de arquivos ou respostas em formato JSON ou XML (*Extensible Markup Language*) (KELLY, 2013).

É esperado com a adoção do formato HAL, uma comunicação simplificada e clara entre os agentes produtores e consumidores de nossa API RESTful, uma vez que por meio de atributos e hiperlinks será possível explorar suas funcionalidades, assim como seus sub-recursos associados.

2.6 Tratamento e refino dos dados brutos e a modelagem das informações

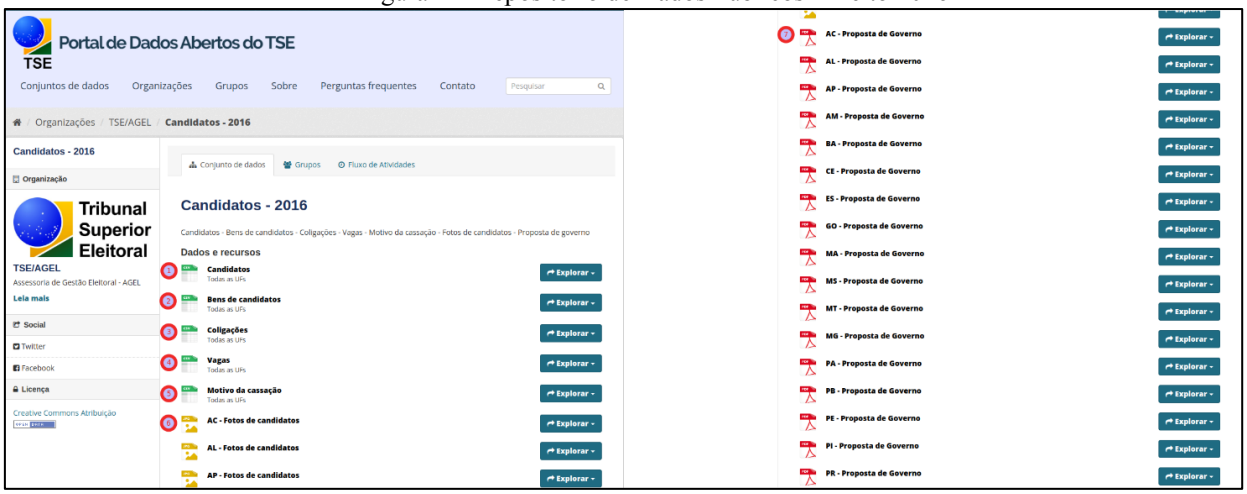
Optou-se pela obtenção dos dados junto ao repositório do TSE (Tribunal Superior Eleitoral)[3], no qual encontram-se disponíveis dados brutos de diversos pleitos nacionais desde 1945, com o propósito de servir a sociedade para o estudo e análise dos processos eleitorais

[3] Disponível em: https://dadosabertos.tse.jus.br/dataset/candidatos-2016

realizados (TSE, 2022). Os dados brutos foram efetivamente obtidos no dia 27/01/2022, relativos ao pleito eleitoral de 2016, já que os mesmos se encontram consolidados.

Desta forma chegou-se a 7 (sete) categorias, conjuntos de dados, ou datasets que serão a base do projeto, sendo eles devidamente numerados: (1) Candidatos, (2) Bens de Candidatos, (3) Coligações, (4) Vagas, (5) Motivos de Cassação, (6) Propostas de Governo e (7) Fotos de Candidatos, conforme apresentado na Figura 2. Sendo selecionadas somente as 5 (cinco) primeiras destas, as quais possuem dados relevantes ao projeto.

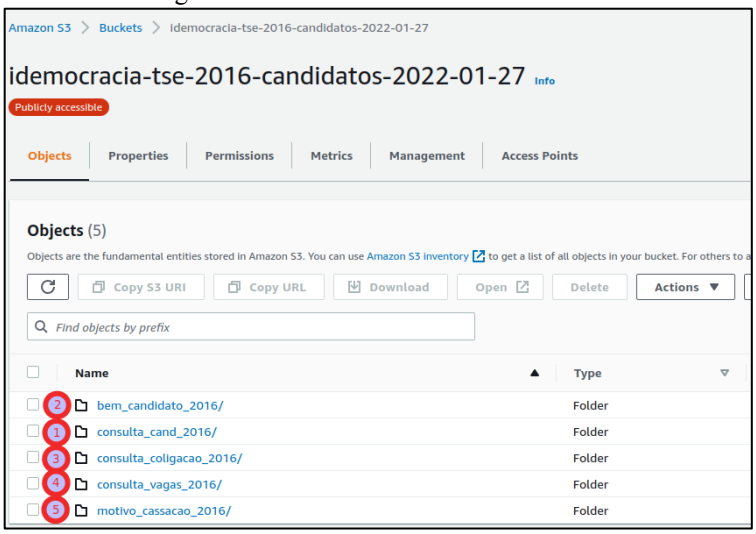
Figura 2 – Repositório de Dados Públicos - Pleito 2016



Fonte: TSE

O formato dos dados originalmente encontrava-se junto a arquivos compactados, sendo por sua vez estes descompactados e preservados em uma estrutura paralela própria, definida como *data lake*, por meio do serviço *AWS S3*, totalizando 8.2 GB de armazenamento, distribuídos em 15098 (quinze mil e noventa e oito) arquivos, em uma estrutura organizacional análoga a utilizada pelo TSE, por meio de 5 (cinco) *bukets*, que são estruturas similares a pastas, sendo eles devidamente numerados: (1) consulta_cand_2016, (2) bem_candidato_2016, (3) consulta_coligacao_2016, (4) consulta_vagas_2016 e (5) motivo_cassacao_2016, acessível por meio de endereço eletrônico[4], apresentada na Figura 3.

Figura 3 – *Data Lake – AWS* - Pleito 2016



Fonte: Elaborada pelo autor

[4] Disponível em: https://idemocracia-tse-2016-candidatos-2022-01-27.s3.amazonaws.com

Tal ação teve por objetivos, preservar o estado dos dados na forma em que se encontravam na data de sua obtenção, possibilitando assim seu versionamento, bem como ser a fonte de consumo dos processos de ETLs que viriam a ser desenvolvidos e também se tornar ponto de auditoria das informações posteriormente tratadas, enriquecidas e distribuídas junto a API RESTful.

Entretanto, em uma análise preliminar detectou-se que os dados em sua grande parte se encontravam em formato CSV (*Comma-separated values*), ou em tradução livre, valores separados por vírgula, sendo estes, arquivos de texto plano simples, onde os valores encontram-se delimitados pelo caractere por vírgula, podendo esta separação ser considerada como uma divisão de colunas, tal qual em programas de planilhas eletrônicas.

Sendo as 5 (cinco) categorias previamente definidas contam com um padrão próprio para seus arquivos em formato CSV, sendo tal padrão definido e descrito pelo próprio TSE em um arquivo único disponibilizado junto a cada *bucket*, ou pasta de cada categoria, sob a alcunha de “leiame.pdf”.

Após a análise exploratória dos dados brutos, iniciou-se o desenvolvimento da modelagem da base de dados relacional, que posteriormente seria alimentada pelos processos de ETL. Desta forma foi desenvolvido o ERD (*Entity Relationship Diagram*), ou DER (Diagrama Entidade Relacionamento)[5], que tem como proposito descrever de maneira abstrata um determinado domínio por meio de suas entidades e relacionamentos (CHEN, 2002).

O domínio do projeto foi modelado inicialmente a partir da criação de um esquema, ou *schema*, com o objetivo de promover uma separação lógica entre as entidades ou tabelas pertencentes ao modelo, sendo assim denominado “tse” pois, caso outras fontes de dados venham a ser adicionadas ao projeto, estas por sua vez também receberão *schemas* próprios.

Durante o processo de análise dos dados brutos, também foram identificadas diversas inconformidades junto às informações, tais como palavras e nomes separados por mais de um espaço em branco, a presença de caracteres especiais, a mescla entre letras minúsculas e maiúsculas, assim como informações numéricas onde se havia expectativa de somente caracteres textuais, sendo assim se fez necessário uma etapa de pré-processamento, materializado sob a função “tratamento_data_frame” (Figura 4), a qual por meio da biblioteca PANDAS foi executada em todo o conjunto de dados, promovendo desta forma o processo para sua higienização e normatização (Figura 5).

Figura 4 - Função – tratamento_data_frame (_script_candidaturas.py)

```
#
def tratamento_data_frame(data_frame):

        # Removendo multiplos espaços em branco.
        data_frame = data_frame.replace({'\s+': ' '}, regex=True)

        # Removendo caracter especial "¨".
        for column in data_frame:
                data_frame[column] = data_frame[column].apply(lambda x : x.replace(u"\u00A8", " ") if isinstance(x, str) else x)

        # ASCI II.
        for column in data_frame:
                data_frame[column] = data_frame[column].apply(lambda x : unidecode.unidecode(x) if isinstance(x, str) else x)

        # Upper.
        for column in data_frame:
                data_frame[column] = data_frame[column].apply(lambda x : x.upper() if isinstance(x, str) else x)

        # Trim.
        for column in data_frame:
                data_frame[column] = data_frame[column].apply(lambda x : x.strip() if isinstance(x, str) else x)

        # String.
        for column in data_frame:
                data_frame[column] = data_frame[column].astype(str)

        return data_frame
```

Fonte: Elaborada pelo autor

[5] Disponível em: https://github.com/limiardigital/idemocracia/tree/main/src/erd

Figura 5 – Aplicação função – tratamento_data_frame (_script_candidaturas.py)

```
228             #
229             data_frame = pd.read_csv(arquivo, encoding='iso-8859-1', delimiter=";", header=0)
230
231             #
232             data_frame = tratamento_data_frame(data_frame)
```

Fonte: Elaborada pelo autor

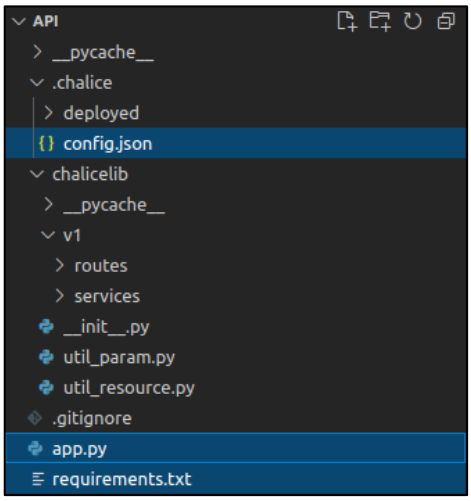
Desta forma o script ETL conta com cerca de 1667 (um mil e seiscentos e sessenta e sete) linhas, disponível junto ao repositório público do projeto[6], sendo dividido em 5 (cinco) grandes segmentos lógicos, os quais fazem o processamento sequencial dos 5 (cinco) conjuntos de dados inicialmente elencados e obtidos junto ao TSE e agora preservados em um *data lake*, sendo eles: Coligações, Candidatos, Bens de Candidatos, Motivos de Cassação e Vagas.

2.7 Desenvolvimento e documentação da API RESTful

Concluídas as etapas de análise, modelagem e ETL, deu-se início à última etapa do projeto, sendo o desenvolvimento e documentação da API RESTful. Para essa etapa foi escolhida a arquitetura de microsserviços baseada no serviço *AWS Lambda*, por meio do framework de desenvolvimento *AWS Chalice*, sendo este selecionado devido a sua integração junto aos serviços da AWS bem como sua linguagem de desenvolvimento, também baseado em *Python*.

O projeto foi desenvolvido sob o diretório "/src/api" o qual possui sua estrutura disposta da seguinte forma (Figura 6).

Figura 6 – Estrutura base projeto – AWS Chalice



Fonte: Elaborada pelo autor

Com especial atenção aos arquivos "app.py", "config.json" e "requirements.txt", sendo o primeiro responsável por inicializar e orquestrar a aplicação, o segundo por prover a configuração básica para execução da mesma e o terceiro e ultimo, trata-se de uma boa prática no desenvolvimento de aplicações *Python*, sendo este um arquivo para o detalhamento e

[6] Disponível em: https://github.com/limiardigital/idemocracia/tree/main/src/etl

importação de dependências. No contexto do framework *AWS Chalice* tem especial importância quanto ao processo de publicação do projeto junto ao serviço *AWS Lambda*.

Foram estabelecidas para a API RESTful ao todo 12 (doze) *resources* ou recursos a serem disponibilizados, relativos as 18 (dezoito) entidades inicialmente modeladas junto ao diagrama ERD, com ressalva a 6 (seis) destas, por se tratarem de entidades de ligação, ou complementares, não se fazem necessárias como recurso próprio em seu senso estrito, porém são essenciais como auxiliares no fornecimento de informações e disponibilização de *sub-resources* ou sub-recursos a API (Quadro 2).

Quadro 2 – Relação de Recursos e Entidades

ERD **Esquema / Tabela ou Entidade**	**Recurso API**
tse / pessoa_fisica	https://api.idemocracia.digital/v1/tse/pessoasFisicas
tse / candidatura	https://api.idemocracia.digital/v1/tse/candidaturas
tse / candidatura_bem	https://api.idemocracia.digital/v1/tse/bens
tse / candidatura_motivo_cassacao	Não se aplica, informação complementar ao recurso "candidaturas"
tse / cargo	https://api.idemocracia.digital/v1/tse/cargos
tse / partido	https://api.idemocracia.digital/v1/tse/partidos
tse / coligacao_partidaria	https://api.idemocracia.digital/v1/tse/coligacoesPartidarias
tse / coligacao_partidaria_partido	Não se aplica, informação complementar aos recursos "partidos" e "coligacoesPartidarias"
tse / motivo_cassacao	Não se aplica, informação complementar ao recurso "candidaturas"
tse / pleito_regional	https://api.idemocracia.digital/v1/tse/pleitosRegionais
tse / pleito_regional_cargo	Não se aplica, informação complementar aos recursos "pleitosRegionais" e "cargos"
tse / pleito_geral	https://api.idemocracia.digital/v1/tse/pleitosGerais
tse / pleito_geral_cargo	Não se aplica, informação complementar aos recursos "pleitosGerais" e "cargos"
tse / pais	https://api.idemocracia.digital/v1/tse/paises
tse / unidade_federativa	https://api.idemocracia.digital/v1/tse/unidadesFederativas
tse / municipio	https://api.idemocracia.digital/v1/tse/municipios
tse / fonte	https://api.idemocracia.digital/v1/tse/fontes
tse / fonte_referencia	Não se aplica, informação complementar ao recurso "fontes"

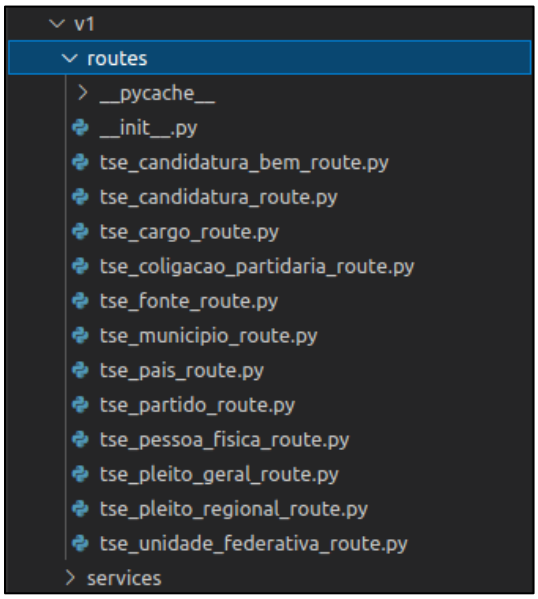
Fonte: Elaborada pelo autor

A nomenclatura dos recursos foi definida na forma plural devido às boas práticas para o desenvolvimento de APIs REST, assim como seus sub-recursos, que são expostos a partir da própria API por meio da implementação do mecanismo de HATEAOS e HAL aplicados ao projeto.

O projeto *AWS Chalice* encontra-se dividido em 2 (duas) camadas, "*routes*" ou rotas e "*services*" ou serviços, com o objetivo de promover uma melhor organização e coesão entre seus artefatos, assim como realizar o versionamento e isolamento de nossa API REST, já que está se encontra em sua versão 1, aqui denominada "v1".

Assim sendo, a camada denominada "*v1/routes*" (Figura 7), possui os arquivos *Python* responsáveis por expor e controlar junto aos usuários o acesso HTTP aos recursos da API REST, por meio de suas operações ou verbos e códigos de status.

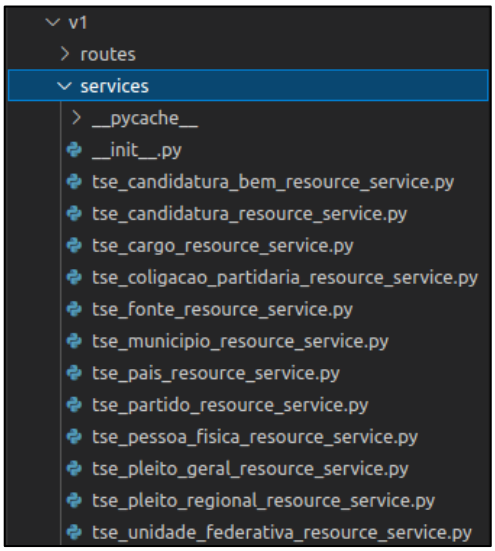
Figura 7 - "v1/routes" do projeto - AWS Chalice



Fonte: Elaborada pelo autor

Uma vez realizada a requisição HTTP junto ao recurso e seu pré-processamento e validação executados pela camada "*routes*", a mesma é delegada e processada pela camada denominada "*services*" (Figura 8), a qual possui os arquivos *Python* responsáveis pelas regras de negócio, comunicação junto à base de dados e processamento dos registros e seu retorno em formato JSON.

Figura 8 - "services" do projeto - AWS Chalice



Fonte: Elaborada pelo autor

Entretanto, buscando simplificar as etapas de pós-processamento dos registros obtidos junto à base de dados, a partir da transformação dos mesmos em registros no formato JSON, foi feito uso de funções nativas ao SGDBR PostgreSQL direcionadas a este fim, por meio das rotinas "json_build_object", "json_agg" e "json_build_array" (JSON, 2022), promovendo uma maior assertividade e produtividade no desenvolvimento dos recursos a serem providos pela API.

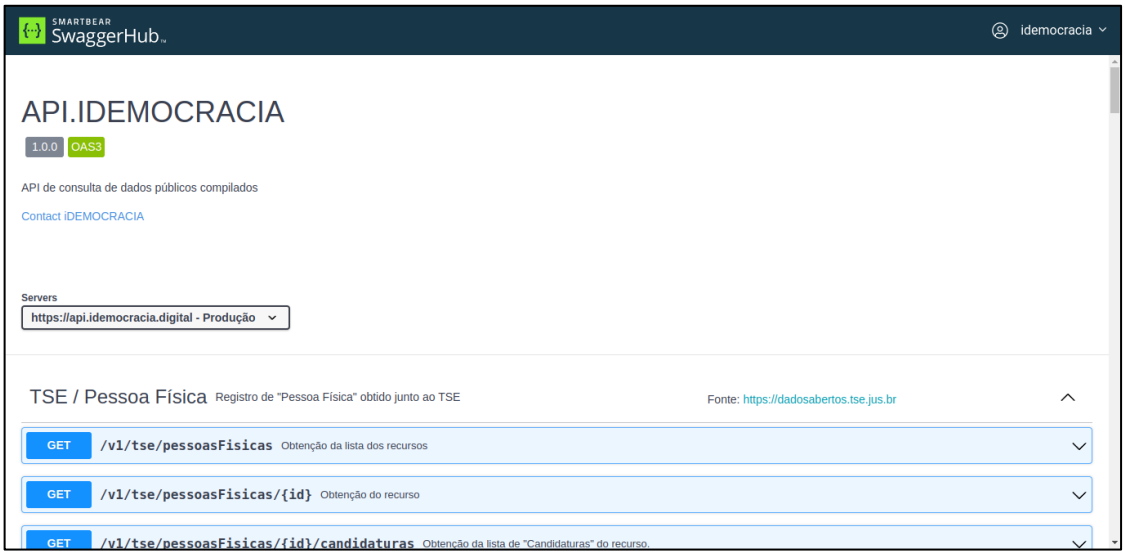
Sob a mesma ótica foi implementada também por meio de funções nativas ao SGDBR PostgreSQL, as funcionalidades relativas a paginação dos resultados foi alcançada por meio das rotinas "*limit*" e "*offset*" (LIMIT, 2022), reduzindo de maneira significativa o uso de recursos por parte da camada "*services*" e junto a base de dados, uma vez que os dados somente são obtidos e processados conforme sejam estritamente necessários.

Os mesmos princípios se aplicam aos demais 12 (doze) recursos desenvolvidos junto ao projeto *AWS Chalice*, sendo realizado posteriormente sua publicação junto à plataforma AWS e seus demais pormenores e configurações realizadas de maneira que seja possível o acesso aos usuários a todos os serviços e recursos disponibilizados pela API RESTful.

Em última instância é necessário o desenvolvimento de sua documentação para os usuários finais, os quais farão uso efetivo das informações e devem possuir uma interface de fácil acesso e clara compreensão, assim como todas as possibilidades disponibilizados pela API.

Desta forma optou-se pela utilização do padrão *OPEN API* em sua versão 3 (três) para composição de tal documentação, em conjunto a ferramenta *SWAGGER HUB* para publicação, exposição e disponibilização de um ambiente de testes seguro (Figura 9), acessível por um endereço eletrônico[7].

Figura 9 – Documentação OPEN API e SWAGGER HUB



Fonte: Elaborada pelo autor

Além disso, também foi desenvolvido também um repositório público para o projeto, junto ao serviço *GITHUB*, sob alcunha de "idemocracia", disponível para análise e colaboração da comunidade, sob a licença MIT de código aberto, onde encontram-se todos os arquivos e artefatos desenvolvidos e aqui descritos[8].

[7] Disponível em: https://app.swaggerhub.com/apis-docs/LIMIAR.DIGITAL/iDEMOCRACIA/1.0.0
[8] Disponível em: https://github.com/limiardigital/idemocracia

3 CONSIDERAÇÕES FINAIS

Este artigo teve como objetivo realizar a proposição de arquitetura para o desenvolvimento de um produto de dados e seus principais elementos, desafios e soluções, o qual foi obtido êxito por meio da publicação do projeto "iDEMOCRACIA", sob uma licença de código aberto, possibilitando agregar fontes de dados públicas diversas, distribuídas em formatos heterogêneos.

Sendo este último ponto, o maior desafio do projeto, já que a diversidade de formatos implica na elaboração de processos de ETLs únicos para cada fonte de dados, sendo necessário um grande esforço de análise e desenvolvimento, já que estes não podem ser reutilizados, pois o contexto de cada qual está intrinsicamente relacionado a sua origem, formato e meio de publicação.

Embora tal cenário fosse esperado, por se tratar de uma solução baseada em *Big Data*, e mesmo que seu objetivo primário tenha sido alcançado, ainda resta como perspectiva futura agregar o máximo de fontes de dados públicos providas pelo Estado brasileiro, bem como fomentar o desenvolvimento de soluções derivadas ou baseadas sob estes princípios, democratizando o acesso a informação a todos os brasileiros, para que possam tomar suas decisões de maneira clara e consciente.

Por este motivo se fazem ainda mais necessárias iniciativas como as do projeto "iDEMOCRACIA", coletando, preservando e divulgando informações de interesse público, de maneira isonômica e apartidária.

REFERÊNCIAS

AWS. **Amazon Global Infrastruture**. 2022. Disponível em: https://aws.amazon.com/pt/about-aws/global-infrastructure. Acesso em: 03. mai. 2022.

AWS ALB. **O que é Application Load Balancer**. 2022. Disponível em: https://docs.aws.amazon.com/pt_br/elasticloadbalancing/latest/application/introduction.html. Acesso em: 01. mai. 2022.

AWS API GATEWAY. **Amazon API Gateway**. 2022. Disponível em: https://aws.amazon.com/pt/api-gateway. Acesso em: 01. abr. 2022.

AWS CHALICE. **CHALICE**. 2022. Disponível em: https://aws.github.io/chalice. Acesso em: 26. abr. 2022.

AWS EC2. **Amazon EC2**. 2022. Disponível em: https://aws.amazon.com/pt/ec2. Acesso em: 03. mai. 2022.

AWS LAMBDA. **AWS Lambda**. 2022. Disponível em: https://aws.amazon.com/pt/lambda>. Acesso em: 20. abr. 2022.

AWS RDS. **AWS RDS**. 2022. Disponível em: https://aws.amazon.com/pt/rds. Acesso em: 27. abr. 2022.

AWS ROUTE 53. **Amazon Route 53**. 2022. Disponível em: https://aws.amazon.com/pt/route53. Acesso em: 03. mai. 2022.

AWS S3. **Amazon S3**. 2022. Disponível em: https://aws.amazon.com/pt/s3. Acesso em: 03. mai. 2022.

CHEN, Perter P. **Entity-Relationship Modeling: Historical Events, Future Trends, and Lessons Learned.** 2002. Disponível em: http://bit.csc.lsu.edu/~chen/pdf/Chen_Pioneers.pdf. Acesso em: 18. out. 2022.

DATACAMP. **Top programming languages for data scientists in 2022**. 2022. Disponível em: https://www.datacamp.com/blog/top-programming-languages-for-data-scientists-in-2022. Acesso em: 13. mai. 2022.

FIELDING, Roy Thomas. **Architectural Styles and the Design of Network-based Software Architectures**. 2000. Disponível em: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf. Acesso em: 18. abr. 2022.

GUPTA, Lokesh. **HATEOAS Driven REST APIs,** 2021. Disponível **e**m: https://restfulapi.net/hateoas. Acesso em: 18. out. 2022.

GUPTA, Lokesh. **REST Architectural Constraints**. 2022. Disponível em: https://restfulapi.net/rest-architectural-constraints. Acesso em: 18. abr. 2022.

IBM. **REST APIs**. 2022. Disponível em: https://www.ibm.com/cloud/learn/rest-apis. Acesso em: 18. abr. 2022.

JSON. **JSON Functions and Operators**. 2022. Disponível em: https://www.postgresql.org/docs/12/functions-json.html. Acesso em: 13. jun. 2022.

KELLY, Mike, **HAL - Hypertext Application Language,** 2013. Disponível em: https://stateless.group/hal_specification.html. Acesso em: 18. abr. 2022.

LIMA, Guilherme. **REST**. 2020. Disponível em: https://www.alura.com.br/artigos/rest-conceito-e-fundamentos. Acesso em: 18. out. 2022.

LIMIT. **LIMIT and OFFSET**. 2022. Disponível em: https://www.postgresql.org/docs/12/queries-limit.html. Acesso em: 13. jun. 2022.

OPEN API. **OPEN API INITIATIVE**. 2022. Disponível em: https://www.openapis.org/about. Acesso em: 01. abr. 2022.

POSTGRESQL, **POSTGRESQL**. 2022. Disponível em: https://www.postgresql.org/about. Acesso em: 07. mai. 2022.

RESTFUL API, **What is REST**. 2022. Disponível em: https://restfulapi.net. Acesso em: 18. mai. 2022.

ROCHA, Cristina T. da Costa; BASTOS, João Augusto de Souza Leão A.; CARVALHO, Marília Gomes de. **ASPECTOS DA SOCIEDADE EM REDE NA ERA DA INFORMAÇÃO**. 2020. Disponível em: http://revistas.utfpr.edu.br/pb/index.php/revedutec-ct/article/view/1074. Acesso em: 18. out. 2022.

VESTER, John. **Exploring the API-First Design Pattern**. 2022. Disponível em: https://dzone.com/articles/exploring-the-api-first-design-pattern. Acesso em: 03. mai. 2022.

TSE. **Repositório de dados eleitorais**. 2022. Disponível em: https://dadosabertos.tse.jus.br/dataset. Acesso em: 27. jan. 2022.