UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
UNIDADE ACADÊMICA DE SISTEMAS E COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

JOEBERTH AUGUSTO CORDEIRO DE SOUZA

UTILIZANDO PROTOCOL BUFFERS PARA FACILITAR A COLABORAÇÃO NO PROJETO DADOSJUSBR

CAMPINA GRANDE - PB

2021

JOEBERTH AUGUSTO CORDEIRO DE SOUZA

UTILIZANDO PROTOCOL BUFFERS PARA FACILITAR A COLABORAÇÃO NO PROJETO DADOSJUSBR

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Professor Dr. Nazareno Ferreira de Andrade.

CAMPINA GRANDE - PB

2021

S729u Souza, Joeberth Augusto Cordeiro de.
Utilizando protocol buffers para facilitar a colaboração no Projeto DadosJusBR. / Joeberth Augusto Cordeiro de. – 2021.

10 f.

Orientador: Prof. Dr. Nazareno Ferreira de Andrade.
Trabalho de Conclusão de Curso - Artigo (Curso de Bacharelado em Ciência da Computação) - Universidade Federal de Campina Grande; Centro de Engenharia Elétrica e Informática.

1. Dados jurídicos. 2. Transparência de dados. 3. Libertação dos dados. 4. Projeto DadosJusBr. 5. Serialização de dados. 6. Codificação em Python. 7. Protocol buffers. I. Andrade, Nazareno Ferreira de. II. Título.

CDU:004(045)

Elaboração da Ficha Catalográfica:

Johnny Rodrigues Barbosa
Bibliotecário-Documentalista
CRB-15/626

JOEBERTH AUGUSTO CORDEIRO DE SOUZA

UTILIZANDO PROTOCOL BUFFERS PARA FACILITAR A COLABORAÇÃO NO PROJETO DADOSJUSBR

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

Professor Dr. Nazareno Ferreira de Andrade
Orientador – UASC/CEEI/UFCG

Professor Dr. Reinaldo Cézar Morais Gomes
Examinador – UASC/CEEI/UFCG

Professor Dr. Tiago Lima Massoni
Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 20 de outubro de 2021.

CAMPINA GRANDE - PB

ABSTRACT

DadosJusBr is a non-profit project with the objective of presenting in a detailed and consolidated way the remuneration information of the agencies that make up the Brazilian justice system, formed by the Public Prosecutors, Defenders, Attorneys and the Judiciary with the courts and councils, together add up to 156 agencies. This process is called 'Libertação dos dados' and has four stages: Collection, Validation, Packaging and Storage. It is in the collection stage that the growth of the project is associated, as it is necessary to coding the collectors, one for each agency. DadosJusBr is an open source project, so the community can participate, writing collectors in multiple programming languages, such as Go and Python. With the use of another programming language, also encompassing dynamic typing where it is more difficult to force a schema considering the type, several problems arise to restrict the data schema. The main one is consistency in the serialization of collected data, which is very important for storage and transmission between stages, as the standard way that languages serialize data is different. In this work we proposed and implemented the use of Protocol Buffers (PB) to make it easier to maintain, transmit and store data consolidated by DataJusBr. We currently have 52 agencies collected, among them the MPPB, coded in Golang, the website of the National Council of Justice (CNJ), coded in python, which were our data collectors that we used in this work. Adapting crawlers and parsers, changing all fields of these collectors to deal with the new data transmission format, resulted in unexpected difficulties, such as dealing with timestamp between the two languages and transmitting the data in PB in text format, thus achieving the serialization of data at all stages. Thus, consolidating the serialization and transmission of data between collectors of different languages, making DadosJusBr more democratic and comprehensive, facilitating the contribution.

Utilizando Protocol Buffers para facilitar a colaboração no projeto DadosJusBR

Joeberth Souza
Universidade Federal De Campina Grande
Campina Grande, Paraíba, Brasil
joeberth@gmail.com

Daniel Fireman
Instituto Federal De Alagoas
Arapiraca, Alagoas, Brasil
daniel.fireman@ifal.edu.br

Nazareno
Universidade Federal De Campina Grande
Campina Grande, Paraíba, Brasil
nazareno@gmail.com

RESUMO

O DadosJusBr é um projeto sem fins lucrativos com o objetivo de apresentar de forma detalhada e consolidada as informações de remuneração dos órgãos que constituem o sistema de justiça brasileiro, formado pelos Ministérios Públicos, Defensorias, Procuradorias e o Judiciário com os tribunais e conselhos, juntos somam 156 órgãos. Esse processo é chamado de 'Libertação dos dados' e possui quatro estágios: Coleta, Validação, Empacotamento e Armazenamento. É no estágio da coleta que o crescimento do projeto está associado, pois é necessária a codificação dos coletores, um para cada órgão. O DadosJusBr é um projeto de código fonte livre aberto, sendo assim a comunidade pode participar, escrevendo coletores em múltiplas linguagens de programação, como Go e Python. Com o uso de mais uma linguagem de programação, englobando também a tipagem dinâmica onde é mais difícil forçar um esquema considerando o tipo, surgem diversos problemas para restringir o esquema de dados. O principal deles é a consistência na serialização dos dados coletados, que é muito importante para armazenamento e transmissão entre estágios, pois o modo padrão que as linguagens serializam dados é diferente. Neste trabalho propusemos e implementamos a utilização de Protocol Buffers (PB) para tornar mais fácil manter, transmitir e armazenar dados consolidados pelo DadosJusBr. Atualmente temos 52 órgãos coletados, dentre eles o MPPB, codificado em Golang, o site do Conselho Nacional de Justiça (CNJ), codificado em python, que foram os nossos coletores de dados que utilizamos neste trabalho. Adaptar os crawlers e parsers, alterando todos os campos desses coletores para lidar com o novo formato de transmissão de dados, acarretou em dificuldades inesperadas, como lidar com timestamp entre as duas linguagens e transmitir o dado em PB no formato de texto, conseguindo assim a serialização dos dados em todos os estágios.

Assim, consolidando a serialização e transmissão dos dados entre coletores de diferentes linguagens, tornando o DadosJusBr mais democrático e abrangente, facilitando a contribuição.

Palavras-chave

Protocol Buffers, Transparência, Libertação dos dados, Serialização.

1. INTRODUÇÃO

Com o passar dos anos, a quantidade de informações disponíveis tem aumentado, assim como a acessibilidade a essas informações. Somado a isso, em 2011, no Brasil, foi criada a lei de número 12.527, também conhecida como a Lei de Acesso à Informação (LAI), que tem como objetivo exigir que os órgãos integrantes da administração direta dos poderes Executivo, Legislativo e Judiciário disponibilizem suas informações de forma pública e acessível para toda a população brasileira. Hoje, todos os tribunais e ministérios públicos disponibilizam em seus sites informações detalhadas sobre seus gastos. Porém a LAI pouco diz sobre a forma como esses dados devem ser disponibilizados. Por isso, nos sites dos órgãos são encontrados arquivos em diversos formatos (html, pdf, planilhas eletrônicas, json etc). Além disso, nomenclaturas e formatação também se distinguem entre os órgãos, o que torna difícil para o cidadão, realizar um controle social efetivo com base nesses dados. O projeto DadosJusBR surge então com o objetivo de realizar o processo de 'libertação dos dados' de gastos com remuneração do sistema de justiça do Brasil, isto é, apresentar de forma detalhada, organizada e unificada esses dados, promovendo um controle social mais simples sobre os gatos do poder judiciário, Ministério Público, Defensoria Pública e Procuradorias.

Para realizar a libertação dos dados, o processo de libertação dos dados de um determinado mês de um órgão engloba 4 etapas: Coleta, Validação, Empacotamento e Armazenamento. A comunicação entre cada estágio acontece de forma padronizada. Depois da coleta, os estágios recebem como entrada padrão (Standard input) a saída padrão (Standard output) do estágio anterior, sempre em texto, mas no formato de JSON. O crescimento do projeto está associado ao estágio de coleta, onde é necessário codificar coletores, um para cada órgão. Pode-se utilizar qualquer linguagem, embora exista uma preferência de uso das linguagens Go e Python. Embora o uso de mais de uma linguagem de codificação aumente as chances de contribuição para o projeto, traz algumas dificuldades na consolidação dos dados, evidenciadas quando tentamos aplicar a uma linguagem de tipagem dinâmica como Python, um esquema definido com base

no tipo. Uma dessas dificuldades é a serialização dos dados recebidos pelos coletores, pois o modo com que as linguagens serializam os dados é diferente, o que exige tratamentos diferentes para cada linguagem na esteira de estágios do DadosJusBR.

Para solucionar esse problema, propusemos e implementamos o uso do Google Protocol Buffers (PB) no DadosJusBr. PB é um método de serialização de dados estruturados de forma binária, uma solução robusta, mantida pela Google e utilizada por grandes empresas. O PB permite que a equipe do DadosJusBr descreva o esquema de dados emitidos pelos coletores usando a linguagem de propósito específico que permite descrever estruturas de dados. A partir dessa descrição agnóstica de linguagem, as estruturas de dados são compiladas nas linguagens nativas de código, no nosso caso, principalmente em Gol e Python, e consolidam o esquema cross-language.

2. MOTIVAÇÃO

O DadosJusBR é um projeto sem fins lucrativos que seu crescimento está associado à participação direta da comunidade na codificação de coletores. Para se tornar mais abrangente e permitir que desenvolvedores com diferentes backgrounds colaborem, é possível colaborar em qualquer linguagem, as mais comuns são Go e Python. O uso de mais de uma linguagem acarreta uma série de problemas, pois o modo que as linguagens lidam com a serialização de dados é diferente. Python é uma linguagem de tipagem dinâmica, isto é, tem a habilidade de escolher o tipo do dado de acordo com o valor atribuído à variável em tempo de execução dinamicamente, essa característica da linguagem, faz com que fique mais difícil forçar um esquema baseado no tipo, isso corrobora para o problema na transmissão e armazenamento dos dados.

Uma forma de ilustrar essa falta de consistência na serialização é observar a codificação dos coletores em python. O resultado da coleta segue um esquema de dados que possui um campo chamado ‘timestamp’, o qual corresponde a uma marcação temporal de quando o evento ocorreu, no caso em questão, a raspagem dos dados do determinado órgão. O estágio da validação, é codificado em golang e recebe os dados dos coletores no formato de JSON, ao realizar a leitura desses dados, a validação encontra um erro de tipo no campo ‘timestamp’, a forma com que Golang trata esse dado de tempo, é diferente da forma de Python, uma solução alternativa foi aplicada nos coletores em python, porém não resolve o problema de forma genérica para qualquer linguagem e campo de dado. Então, surge a oportunidade de utilizar o Protocol Buffers para descrever o esquema e garantir a integridade e transmissão dos dados em qualquer linguagem.

3. BACKGROUND TEÓRICO

3.1 Protocol Buffers (PB)

Protocol Buffers é um formato de dados binários bastante usado na serialização e transmissão de dados através da rede. Foi desenvolvido pela Google inicialmente para lidar com um problema interno de grande número de requisições e respostas no servidor. PB introduziu a compatibilidade do esquema de dados em diferente versões de projeto, isso porque, enquanto o modelo antigo de lidar com os dados suportava apenas uma versão e necessitava de ajuste a cada mudança no esquema, PB por outro lado ignora novos campos de dados não suportados, ou seja, podendo ser utilizado por diferentes releases do projeto. PB também é utilizado quando é necessário a troca de uma grande quantidade de dados em um curto tempo. Assim como XML e JSON, outros dois formatos de serialização de dados, PB se destaca por ser mais simples e mais rápido do que os concorrentes, principalmente quando o alvo são sistemas em larga escala.

PB permite criar arquivos proto descrevendo o esquema dos dados que serão enviados, esses arquivos são compostos por ‘messages’ que descrevem cada parte do esquema (Fig 1). Os arquivos proto são compilados para código em linguagens de programação específicas, como Python ou Java, fornecendo classes de acesso aos dados de acordo com a linguagem de programação que vai ser utilizada. Ou seja, PB permite gerar o mesmo esquema de dados em diferentes linguagens de forma padronizada, sem que seja necessário conhecer a linguagem alvo de conversão. Essa facilidade de comunicação de dados entre linguagens torna o PB bastante popular na comunidade.

```
syntax = "proto3";

message SearchRequest {
  string query = 1;
  int32 page_number = 2;
  int32 result_per_page = 3;
}
```

Figura 1 - Exemplo de message PB

3.2 DadosjusBR

A Lei de Acesso à Informação (Lei n. 12.527, de 2011), regula a obrigatoriedade da disponibilização na internet das informações acerca dos gastos públicos, incluindo a folha de pagamento. Porém, a lei não é específica com relação a padronização e assim, órgãos com a mesma estrutura remuneratória acabam disponibilizando informações de diversas maneiras, inclusive utilizando formatos diferentes (PDF, HTML, ODS, XLS, JSON e etc), muitos dos quais com acesso difícil e formato não amigáveis à ferramentas de análise e processamento de dados.

Inspirados em projetos como o Brasil.io[1][1] e Querido Diário[2][2], o projeto DadosJusBr[3] tem como principal objetivo denunciar as dificuldades que os cidadãos têm para conseguir acessar os dados sobre a folha de pagamento dos órgãos do sistema de justiça (MPs e TJs) e prover acesso a essas informações de forma amigável e em formato aberto.

A coleta de dados no DadosJusBR é dividida em um pipeline com 4 etapas. Este é uma sequência de estágios que visam realizar uma tarefa macro, onde essa tarefa foi dividida em uma série de programas executados utilizando o docker[3]. Docker é uma ferramenta que nos permite abstrair os detalhes de

[1] https://brasil.io/home/

[2] https://queridodiario.ok.org.br/

[3] https://www.docker.com/

implementação, trazendo o benefício que nós não precisamos saber em qual linguagem foi feito ou quais são as dependências do seu código. No nosso contexto, todos os coletores e os demais estágios são executados da mesma forma independente se foram codificados em Go ou Python, isso é possível porque possuem um *Dockerfile,* arquivo que abstrai como o programa deve ser executado, com o script da execução, assim, com o uso dos comandos '*docker build*' para fazer a construção da imagem contendo as dependências necessárias para execução e o comando '*docker run'* para realizar a execução, assim tornando o processo agnóstico a linguagem, ambiente e dependências. O tratamento de erros, fica por conta de um estágio especial que chamamos de *ErrorHandler*. Ele será construído e executado como os demais, porém, se ocorrer outro erro, interrompemos a execução e retornamos todos os detalhes da execução do pipeline até aquele ponto. Considerando erro quando a construção ou execução de uma imagem levanta um erro durante seu processamento ou quando não levanta erro mas retorna um status diferente de 0(OK). De forma simplificada, a saída padrão de um estágio vira a entrada padrão do estágio seguinte, a figura 2 ilustra o processo de execução para um mês do dadosjusBR dividida em suas 4 etapas, que são:

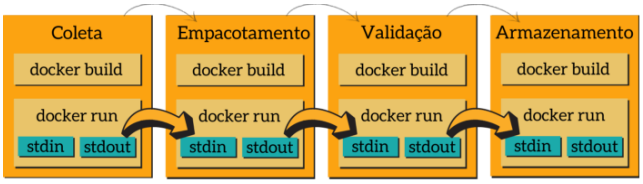

Figura 2 - Pipeline de execução DadosjusBR

3.2.1 Coleta

Coletores são programas que realizam a coleta, conversão e consolidação das folhas de pagamento dos órgãos do sistema de justiça. Hoje possuímos mais de 50 coletores disponibilizados e mais de 2000 meses de dados libertados.

3.2.2 Empacotamento

O empacotador é responsável por criar pacotes em formato aberto com os dados oriundos das coletas e realizar a abertura, (i. e. dados são abertos quando qualquer pessoa pode livremente acessá-los, utilizá-los, modificá-los e compartilhá-los para qualquer finalidade, estando sujeito a, no máximo, a exigências que visem preservar sua proveniência e sua abertura)[4]. Agora, o empacotador recebe os dados do coletor na entrada padrão, gera as planilhas em csv dos dados coletados e consolida em um arquivo compactado essas planilhas, assim como o contrato do esquema desses dados, o qual chamamos de '*datapackage*'. Ao chegar o fim desta etapa, temos um arquivo compactado e passamos os dados para o estágio seguinte.

3.2.3 Validação

O estágio de validação, verifica se todos os arquivos que existem no arquivo compactado que foi gerado na etapa anterior, seguem em sua totalidade, as definições existentes no contrato, que é um descritor frictionless que cria metadados para os arquivos de dados. Ter metadados é importante porque os arquivos de dados sozinhos geralmente não fornecem informações suficientes para compreender totalmente os dados, com isso também é possível adicionar restrições aos dados e com isso garantir a consistência, após a confirmação que os dados obedecem o contrato, a etapa de armazenamento é chamada.

3.2.4 Armazenamento

A última etapa do pipeline, é responsável por receber os dados e arquivos gerados pela etapa anterior e realizar o armazenamento no nosso banco de dados. Os arquivos são armazenados na cloud e retornam um link para realizar o download que é armazenado junto com os dados em formato de documento no banco de dados não-relacional.

4. SOLUÇÃO IMPLANTADA

Utilizamos PB para definir uma interface descritiva do esquema de dados do DadosJusBR, garantindo a consistência da definição e serialização entre múltiplas linguagens independente da tipagem.

4.1 Descrevendo os protos

Para conseguir representar o esquema de dados necessário na execução, tivemos que construir três arquivos proto com o intuito de isolar a responsabilidade de cada esquema de dados e cobrir o fluxo completo do DadosJusBR. Assim com objetivos diferentes surgiram o *coleta.proto*, arquivo que é responsável por traduzir o esquema de dados utilizado na coleta, o *pacote.proto* tem o objetivo de carregar o esquema de dados necessários para gerar as planilhas em csv com os dados obtidos dos coletores, já o *pipeline.proto* tem a função de guardar as estruturas comuns do pipeline DadosjusBR utilizadas para transmitir dados entre os estágios.

Na figura 3, podemos analisar uma mensagem do arquivo *coleta.proto*, que traduz o resultado da coleta em mensagens aninhadas.

```
message ResultadoColeta {
    Coleta coleta = 1;
    FolhaDePagamento folha = 2;
    ProcInfo procinfo = 3;
    Metadados metadados = 4;
}
```

Figura 3 - Exemplo de mensagem de coleta.proto

Na figura 4, analisamos uma mensagem do arquivo *pacote.proto*. que carrega as estruturas necessárias para construção dos arquivos de dados que vão dentro do *Datapackage.*

```
message ResultadoColeta_CSV {
    Coleta_CSV coleta = 1;
    Remuneracoes_CSV remuneracoes = 2;
    FolhaDePagamento_CSV folha = 3;
}
```

Figura 4 - Exemplo de mensagem de *pacote.proto*

Na figura 5, representamos nosso último proto, do pipeline.proto que corresponde a estrutura final usada para transmitir os dados entre os estágios.

```
message ResultadoExecucao {
        ResultadoEmpacotamento pr = 1;
        ResultadoColeta rc  = 2;
}
```

Figura 5 - Exemplo de mensagem de pipeline.proto

4.2 Gerando as classes

Com os protos definidos, é a hora de gerar os arquivos para cada linguagem que vai lidar com as estruturas. No DadosjusBR, atualmente utilizamos duas linguagens de programação, Go e Python. Em Python, o stub gerado são classes e em Go structs. Na figura 6, podemos enxergar a estrutura do repositório de coleta, onde os arquivos *coleta.pb.go* é um stub gerado com o esquema de dados para golang e *coleta_pb2.py*, o mesmo modelo para Python.

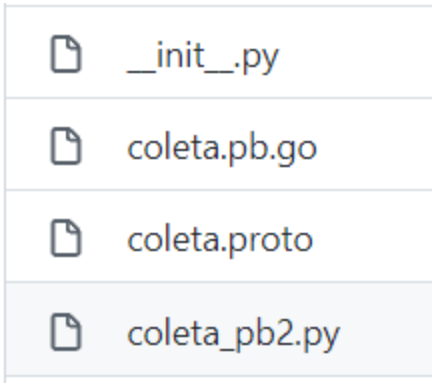

Figura 6 - Arquivos do repositório de coleta

Na figura 7 e na figura 8, podemos ver o resultado da mesma estrutura, Resultado Coleta, após ter sido gerada em go e em python.

```
type ResultadoColeta struct {
    state         protoimpl.MessageState
    sizeCache     protoimpl.SizeCache
    unknownFields protoimpl.UnknownFields

    Coleta    *Coleta           `protobuf:"bytes,1,opt
    Folha     *FolhaDePagamento `protobuf:"bytes,2,opt
    Procinfo *ProcInfo         `protobuf:"bytes,3,opt
}
```

Figura 7 - Estrutura Resultado Coleta em Go.

```
_RESULTADOCOLETA = _descriptor.Descriptor(
  name='ResultadoColeta',
  full_name='ResultadoColeta',
  filename=None,
  file=DESCRIPTOR,
  containing_type=None,
  fields=[
    _descriptor.FieldDescriptor(
      name='coleta', full_name='ResultadoColeta.coleta', index=0,
      number=1, type=11, cpp_type=10, label=1,
      has_default_value=False, default_value=None,
      message_type=None, enum_type=None, containing_type=None,
      is_extension=False, extension_scope=None,
      options=None),
    _descriptor.FieldDescriptor(
      name='folha', full_name='ResultadoColeta.folha', index=1,
      number=2, type=11, cpp_type=10, label=1,
      has_default_value=False, default_value=None,
      message_type=None, enum_type=None, containing_type=None,
      is_extension=False, extension_scope=None,
      options=None),
    _descriptor.FieldDescriptor(
      name='procinfo', full_name='ResultadoColeta.procinfo', index=2,
      number=3, type=11, cpp_type=10, label=1,
      has_default_value=False, default_value=None,
      message_type=None, enum_type=None, containing_type=None,
      is_extension=False, extension_scope=None,
      options=None),
  ],
```

Figura 8 - Estrutura Resultado Coleta em Python.

4.3 Executando o pipeline

O próximo passo, foi adaptar os estágios do DadosjusBR para receber o novo esquema e adaptar dois coletores existentes no DadosjusBR, para o novo formato de transmissão de dados em proto, o do Ministério Público da Paraíba (MPPB), codificado em Golang e o do Conselho Nacional de Justiça (CNJ), codificado em python.

4.3.1 Coletor-MPPB

Por ser ambientado em Go, o coletor conseguiu importar de maneira simples o stub gerado pelo proto, facilitando a adaptação para o esquema de dados baseado em PB e coletando os dados sem maiores problemas.

4.3.2 Coletor-CNJ

Codificado em python, o coletor do CNJ trouxe uma complexidade a mais para a adaptação, isso porque em python, o processo de publicação de dependências é mais complexo do que em go.Python exige que seja realizado o upload do pacote desenvolvido em um repositório de software, utilizamos o *pypi.org,* para fazer o upload do nosso pacote. Após esse

processo, o coletor do cnj conseguiu instalar o pacote e utilizar as classes com o esquema de dados gerados para a linguagem.

4.3.3 Empacotador

Nesse estágio é onde a comunicação entre linguagens é consolidada, os coletores em diferentes linguagens enviam o dado em um formato agnóstico, o empacotador recebe esse dado, importa o pacote proto gerado que traduz os dados para as

planilhas CSV e transmite os dados para o próximo estágio. Nos estágios seguintes, a adaptação ao novo formato de dados aconteceu de maneira mais simples e a transmissão de dados ocorreu sem complicações.

Na figura 9, temos esse processo detalhado.

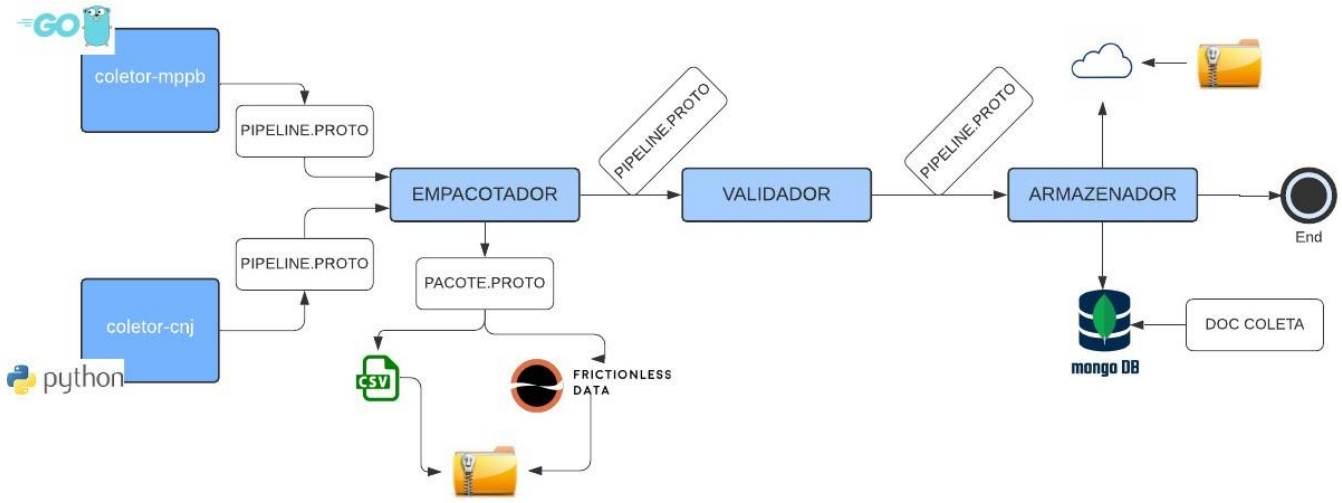

Figura 9 - Fluxograma de execução DadosJusBr

5. DIFICULDADES ENFRENTADAS

O DadosjusBR transmitia os dados no formato JSON entre os estágios, porém esse formato passou a apresentar divergências de serialização, então foi necessário buscar outra alternativa de formato de comunicação entre os estágios. JSON não se apresentou como um formato válido, porque Go e Python divergem ao lidar com o campo *Timestamp*, que é uma marca temporal do evento que ocorreu, na documentação oficial do PB, indica que a saída do campo vai ser uma string e obedecer o formato de data RFC 3339, um exemplo desse formato é "2021-09-20T00:39:04.438Z", que indica que um evento ocorreu no dia 20/09/2021, às 00 horas 39 minutos e 04.438 segundos. Em python, essa saída é respeitada quando convertemos a mensagem do proto para JSON, porém quando geramos o JSON no coletor em golang, vimos que o formato gerado representa a struct de timestamp que PB utiliza, com os campos *seconds e 'nanos'* que representam os segundos e os nanosegundos passados desde 1 de janeiro de 1970. A figura 8 mostra um exemplo do json gerado.

```
"timestamp_coleta": {
  "seconds": 1632098344,
  "nanos": 438708000
},
```

Figura 10 - Timestamp JSON

Essa diferença faz com que o empacotador não consiga desserializar a entrada padrão em uma estrutura válida, o que quebra o fluxo de execução. Para sanar esse impasse, alteramos toda a estrutura de transmissão e passamos a transformar em binário as mensagens e enviar para o próximo estágio.

Após implementar essa mudança, surgiu outro problema, mais uma vez enviar a própria estrutura em binário tem uma diferença entre as linguagens, então precisamos mais uma vez buscar alternativas para solucionar o problema, a solução foi imprimir as mensagens PB em formato de texto. Dessa forma o empacotador passou a conseguir interpretar as entradas padrão independente da linguagem do coletor e o fluxo de execução não teve mais problemas para avançar entre os estágios.

6. IMPLEMENTAÇÃO RESULTANTE

Criamos um repositório *proto*, onde se encontram os arquivos PB com a descrição das estruturas necessárias para a criação dos stubs em python e em go. Para consolidar a efetividade das nossas estruturas, agora em proto, criamos os repositórios *coletor-cnj* e *coletor-mppb*, esses coletores foram responsáveis por adaptar as estruturas dos coletores existentes do CNJ (python) e do MPPB (go) para o novo formato. Os repositórios já existentes *storage, executor, empacotador* e *validador*, também tiveram suas estruturas alteradas para lidar com PB. Os repositórios resultantes são:

github.com/dadosjusbr/proto
github.com/dadosjusbr/coletor-mppb
github.com/dadosjusbr/coletor-cnj
github.com/dadosjusbr/storage
github.com/dadosjusbr/executor
github.com/dadosjusbr/empacotador
github.com/dadosjusbr/validador

7. CONCLUSÃO

A serialização dos dados entre os estágios do DadosJusBR era um problema quando temos coletores escritos em diferentes linguagens, principalmente quando utilizamos linguagens com tipagem dinâmica, isso ocorre porque nessas linguagens é mais difícil fazer com que elas obedeçam um esquema baseado em tipo. A utilização da linguagem Protocol Buffers forneceu a possibilidade de descrever o esquema dos dados que são obtidos pelos coletores e transmitir esses dados em um formato agnóstico à linguagem, sanando problemas de serialização, transmissão e armazenamento. Para fazer isso, esse novo esquema de dados em

PB é compilado em um código-fonte gerado automaticamente na linguagem desejada e utilizá-lo para ler, gravar ou comunicar esses dados de maneira que a continuidade do pipeline do projeto não seja afetado.

Os resultados obtidos foram satisfatórios, hoje é possível desenvolver os coletores em diferentes linguagens sem se preocupar com o aparecimento de problemas na comunicação de dados entre os estágios, essa implementação visa impulsionar o crescimento da computação cívica e tornar o DadosjusBR em um projeto mais democrático e abrangente, ajudando assim a informação sobre o sistema judiciário brasileiro chegar de forma mais clara aos cidadãos.

8. REFERÊNCIAS

[1] Open Knowledge foundation. http://opendefinition.org/.

[2] Querido diário. https://queridodiario.ok.org.br/

[3] Brasil io. https://brasil.io/home/

[4] DadosjusBR. https://dadosjusbr.org/sobre

[5] Pereira, Lorena. DadosJusBR: Executando um Pipeline, https://medium.com/dadosjusbr/dadosjusbr-executando-um-pipeline-cfd26a50165e

[6] Popić, Srđan & Pezer, Dražen & Mrazovac, Bojan & Teslic, Nikola. (2016). Performance evaluation of using Protocol Buffers in the Internet of Things communication. 261-265. 10.1109/SST.2016.7765670.