

UNIVERSIDADE ESTADUAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO E
SISTEMAS

DIEGO PERERA MENDES

**FERRAMENTA DE PRÉ-PROCESSAMENTO E VISUALIZAÇÃO DE DADOS DO
DATASUS**

SÃO LUÍS
2019

DIEGO PEREIRA MENDES

**FERRAMENTA DE PRÉ-PROCESSAMENTO E VISUALIZAÇÃO DE DADOS DO
DATASUS**

Dissertação apresentada ao curso de Mestrado Profissional em Engenharia da Computação e Sistemas na Universidade Estadual do Maranhão como pré-requisito para obtenção do título de Mestre em Engenharia de Computação e Sistemas.

Orientador: Prof. Msc. Antonio Fernando Lavareda Jacob Jr.

SÃO LUÍS
2019

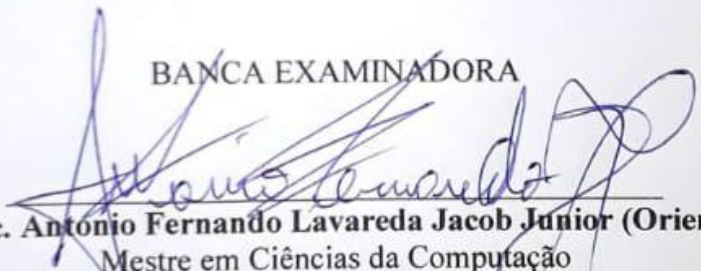
DIEGO PEREIRA MENDES

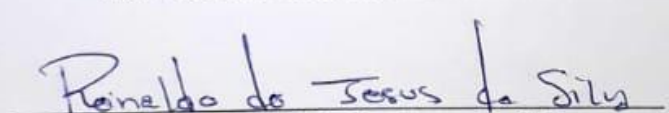
FERRAMENTA DE PRÉ-PROCESSAMENTO E VISUALIZAÇÃO DE DADOS DO DATASUS

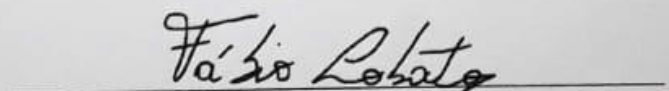
Dissertação apresentada ao curso de Mestrado Profissional em Engenharia da Computação e Sistemas na Universidade Estadual do Maranhão como pré-requisito para obtenção do título de Mestre em Engenharia de Computação e Sistemas sob orientação do Prof. Msc. Antonio Fernando Lavareda Jacob Jr.

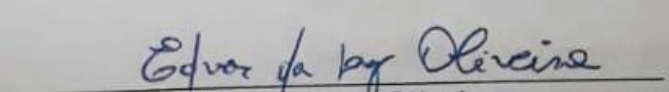
Aprovada em: 05 de fevereiro de 2019

BANCA EXAMINADORA


Prof. Msc. Antonio Fernando Lavareda Jacob Junior (Orientador)
Mestre em Ciências da Computação
Universidade Estadual do Maranhão


Prof. Dr. Reinaldo de Jesus da Silva
Doutor em Informática na Educação
Universidade Estadual do Maranhão


Prof. Dr. Fábio Manoel França Lobato
Doutor em Engenharia Elétrica
Universidade Federal do Oeste do Pará


Edvar da Luz Oliveira
Doutor em Engenharia Elétrica
Universidade Federal Rural da Amazônia



À minha família pela capacidade de acreditar e investir em mim. Mãe, seu cuidado e dedicação foi que deram, em alguns momentos, a esperança para seguir. Pai, sua presença significou segurança e certeza de que não estou sozinho nessa caminhada.

AGRADECIMENTOS

À Deus, que me deu forças e esperança para superar tantos obstáculos e alcançar mais esta meta.

Aos meus pais, Antônio Guimarães Mendes Filho e Nilde Pereira Mendes, por todo o incentivo e esforço depositados durante todos os anos da minha vida, além de compreenderem os dias festivos que tive que abrir mão.

Aos meus irmãos, Tiago Pereira e Matheus Filipe, pela paciência durante noites de estudo.

Aos meus amigos de trabalho que tiraram algumas dúvidas que surgiram durante o trabalho.

À minha esposa Kamyla Costa Pereira, pelo apoio, compreensão e confiança constantes.

Ao meu orientador, Antonio Fernando Lavareda Jacob Junior, pelo apoio, receptividade ao tema, orientação e paciência dispensada, mostrando sempre ser prestativo e interessado em ensinar com paciência.

Aos professores do Departamento de Informática pelos seus ensinamentos e os funcionários do curso, que durante esses anos, contribuíram de algum modo para o crescimento pessoal e profissional.

Aos irmãos do Ministério de Louvor Exaltai que estiveram orando e compreendendo os dias de estudo e me incentivaram.

Aos amigos de faculdade da turma 2016, que sempre estiveram presentes ao meu lado em inúmeros trabalhos e provas nesta difícil caminhada de mestrando, me ensinando muita coisa.

E a todas as pessoas que acreditaram e torceram por mim em todos esses anos de graduação e no que esta conquista traz de positivo à minha vida.

Algumas pessoas acham que foco significa dizer sim para a coisa em que você vai se focar. Mas não é nada disso. Significa dizer não às centenas de outras boas ideias que existem.

(Steve Jobs)

RESUMO

O DATASUS (Departamento de Informática do SUS) fornece informações que apoiam análises objetivas da situação sanitária brasileira, auxiliando em tomadas de decisões baseadas em evidências e elaboração de soluções de ações de saúde. Para acessar tais informações é necessário a utilização das ferramentas disponibilizadas pelo portal do DATASUS que são muito limitadas tecnicamente e complexas. No portal do DATASUS, essas informações estão em arquivos .DBC e criptografadas por meio de um algoritmo privado, dificultando a interpretação das informações. Este trabalho apresenta uma ferramenta desenvolvida em JAVA com o propósito de tratar os dados públicos disponibilizados pelo DATASUS e exportá-los para um banco de dados relacional de uma forma mais simples. Para este fim, foi desenvolvida uma ferramenta responsável pela captura, descriptografia e exportação das informações para um banco de dados relacional.

Palavras-chave: Acesso à Informação; Transparência; SUS; Dados de Saúde Pública; Pré-processamento de dados.

ABSTRACT

DATASUS (Information Technology Department of the SUS) provides information that supports objective analyzes of the Brazilian health situation, assisting in evidence-based decision making and elaboration of health action solutions. In order to access this information, it is necessary to use the tools provided by the DATASUS portal that are very limited technically and complex. In the DATASUS portal, this information is in .DBC files and encrypted by means of a private algorithm, making it difficult to interpret the information. This work presents a tool developed in JAVA with the purpose of treating the public data provided by DATASUS and exporting them to a relational database in a simpler way. To this end, a tool was developed responsible for capturing, decrypting and exporting the information to a relational database.

Keywords: Access to Information; Transparency; SUS; Public health data; Pre-processing of data.

LISTA DE FIGURAS

Figura 1 - JNI (Java Native Interface)	20
Figura 2 - Assinatura do método desejado com modificador NATIVE	20
Figura 3 - Carregando biblioteca nativa para a JVM	20
Figura 4 - Classe teste da calculadora	21
Figura 5 - Utilização da ferramenta JAVAH	21
Figura 6 – Arquivo SomadorJNI.h	21
Figura 7 - Arquivo SomadorJNI.c	22
Figura 8 - Executando código nativo.....	22
Figura 9 - JNA. Fonte: https://www.devmedia.com.br/	23
Figura 10 - Repositório de Dados da CNES.....	26
Figura 11 - Nomenclatura arquivo CNES	27
Figura 12 - Arquitetura ferramenta de pré-processamento e visualização de dados do DATASUS utilizando a biblioteca <i>datasusDbcToDbf.jar</i>	27
Figura 13 – Código da etapa de captura dos dados	28
Figura 14 - JNA aplicado na biblioteca <i>dbcToDbf-1.1.jar</i>	29
Figura 15 - Comandos criação <i>blast_c.dll</i>	30
Figura 16 - Método Java chamando código Nativo em C de conversão DBC em DBF	30
Figura 17 - Trecho de Código em R.....	34
Figura 18 - Trecho de código em JAVA - <i>dbcToDbf-1.1.jar</i>	34
Figura 19 – Comparação de execução usando Arquivo DBC de 194 kb	34
Figura 20 - Comparação de execução usando Arquivo DBC de 938 kb.....	35
Figura 21 - Comparação de execução usando Arquivo DBC de 1.5 Mb	35
Figura 22 - MVC	38
Figura 23 - Arquitetura da aplicação de visualização DATASUS	38
Figura 24 - Diagrama de Entidade e Relacionamento (resumido)	42
Figura 25 - Tela de login	43

Figura 26 - Tela inicial	43
Figura 27 - Tela de importação/transformação dos dados.....	44
Figura 28 - Tela de Relatórios	45
Figura 29 - Resposta média para pergunta 1	49
Figura 30 - Resposta média para pergunta 2.	50
Figura 31 - Resposta média para pergunta 3	50
Figura 32 - Resposta média para pergunta 4	50
Figura 33 - Resposta média para pergunta 5.	51
Figura 34 - Resposta média para pergunta 6	51
Figura 35 - Resposta média para pergunta 7	51
Figura 36 - Resposta média para pergunta 8	52
Figura 37 - Resposta média para pergunta 9	52
Figura 38 - Resposta média para pergunta 10.	52

LISTA DE TABELAS

Tabela 1 - Projetos que utilizam JNA.....	23
Tabela 2 - Dados disponíveis no DATASUS.	25
Tabela 3 - Resultados comparativos da biblioteca JAVA e R.....	36
Tabela 4 - Dicionário de Dados do DER.....	42
Tabela 5 - Rede assistencial de cardiologia sistema SUS	46
Tabela 6 - Cálculo de score para perguntas ímpares	53
Tabela 7 - Cálculo de score para perguntas pares	53

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
CNES	Cadastro Nacional de Estabelecimentos de Saúde
CSV	Comma Separated Values
DATASUS	Departamento de Informática do Sistema Único de Saúde do Brasil.
DCL	Data Compression Library
DER	Diagrama de Entidades e Relacionamentos
DLL	Dynamic-link library
FTP	File Transfer Protocol
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
JNA	Java Native API
JNI	Java Native Interface
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
MVC	Model View Control
PDF	Portable Document Format
PF	Pessoa Física
PJ	Pessoa Jurídica
POJO	Plain Old Java Objects
SQL	Structured Query Language
SUS	Sistema Único de Saúde
TABWIN	Tabulador para Windows
URL	Uniform Resource Locator
XML	Extensible Markup Language

SUMÁRIO

1. INTRODUÇÃO	15
2. FUNDAMENTAÇÃO TEORICA.....	19
2.1. Java Native Interface (JNI)	19
2.2. Java Native Access (JNA)	22
3. TRABALHOS CORRELATOS	24
4. BIBLIOTECA datasusDbcToDbf.jar	25
4.1. Arquitetura da Ferramenta	27
4.1.1.Captura	28
4.1.2.Transformação	29
4.1.3.Visualização de dados	31
4.2. Exemplo de utilização da biblioteca <i>Dbctodbf-1.1.Jar</i>	31
5. ANÁLISE DE DESEMPENHO DA BILIOTECA	33
6. FERRAMENTA DE EXPORTAÇÃO E VISUALIZAÇÃO DE DADOS DO DATASUS	37
6.1. Arquitetura do Sistema	37
6.1.1.Angular JS	39
6.1.2.JAVA.....	39
6.1.3.SPRING.....	39
6.1.4.MyBatis	40
6.1.5.Java JDBC	40
6.2. Modelo de Dados	40
6.3. Modelo de entidade e relacionamento	41
6.4. Interface gráfica	42
6.4.1.Captura e Transformação.....	43
6.4.2.Visualização e Exportação de dados	45
7. RESULTADOS	48
7.1. Resultados do questionário	48
8. CONCLUSÃO	54
APÊNDICE	58
APÊNDICE A – TRECHO DE CÓDIGO EM JAVA COM A UTILIZAÇÃO DA BIBLIOTECA <i>DBCTODBF-1.1.JAR</i>	59
ANEXOS.....	62

ANEXO I - BLAST.C	63
ANEXO II – BLAST.H.....	68
ANEXO III – CERTIFICADO DE REGISTRO DE PROGRAMA DATASUSVIEWER.....	69
ANEXO IV – CERTIFICADO DE REGISTRO DE PROGRAMA DATASUSIMPORT	70

1. INTRODUÇÃO

O Departamento de Informática do SUS – DATASUS concede informações que podem contribuir em estudos e análises objetivas da situação da saúde brasileira, resoluções de problemas baseadas em evidências e elaboração de programas de ações e dados de saúde, (indicadores, informações epidemiológicas e de morbidade, assistência à saúde, estatísticas vitais, informações sobre a rede de assistência à saúde, informações demográficas e socioeconômicas) além de dados financeiros (referentes aos recursos do Fundo Nacional de Saúde transmitidos aos municípios, aos prestadores de serviços de saúde, aos orçamentos públicos de saúde declarados pelos Estados, pelo Distrito Federal e pelos Municípios). Essas bases de dados podem ser consultadas no portal do DATASUS 2018.

É desconfortante a forma como DATASUS disponibiliza as informações da saúde pública brasileira. Grande parte desses dados disponíveis gratuitamente está no formato “.DBC”, uma versão do formato “.DBF” comprimido, que dependem da utilização da ferramenta TABWIN para análise ou conversão para outros formatos, e uma ferramenta de linha de comando chamada DBC2DBF6, limitando-se ao sistema operacional WINDOWS, execução em máquina local do usuário, treinamento específico e o fato de não ser *open source* impossibilitando personalizações de análises/consultas, implementação de melhorias e customizações no *software* de tabulação e na escalabilidade das plataformas de *software*, e quanto ao desenvolvimento de *softwares* derivados, estes seriam vinculados somente a um sistema operacional.

Em meio a esse contexto será apresentado uma nova ferramenta para facilitar/simplificar a captura, interpretação e análises dos dados do DATASUS. Será uma solução *open source*, multiplataforma (sem limitação de sistemas operacionais) e com uma apresentação mais amigável dos dados, facilitando assim sua interpretação e análise para tomadas de decisões.

Vale destacar que as ferramentas *datasusImport* - Sistema de Importação de dados do DATASUS para um banco de dados relacional e *datasusViewer* - Sistema de Visualização de dados do DATASUS advindos de um banco de dados relacional foram registradas juntos ao INPI (Instituto Nacional de Propriedade Intelectual), conforme consta nos certificados do INPI anexos III e IV.

Ao analisar as limitações percebidas nas ferramentas viabilizadas pelo DATASUS, observou-se a necessidade do desenvolvimento de uma ferramenta na linguagem JAVA, que

busque democratizar o acesso aos dados fornecidos pelo DATASUS, capturando as informações via *download*, transformando e descriptografando as informações e exportando os dados para um banco de dados relacional. A ferramenta proposta em questão permitirá uma maior escalabilidade para o desenvolvimento de *softwares* derivados, customizados, personalizados e independente de sistema operacional.

Com a popularização da ferramenta *open source* será permitido que outras pessoas da comunidade científica ou não, que de forma independente, possam desenvolver novas ferramentas de análise de dados, ou aperfeiçoá-los para atender uma necessidade específica.

Podemos citar como exemplos beneficiados com a nova biblioteca:

- Sistemas para indicadores de Saúde;
- Sistemas para avaliar quantificação e qualidade em saúde;
- Sistemas para tomadas de decisão em aplicação de recursos financeiros na saúde;
- Extração de novas informações que são inviáveis devido a forma atual de como os dados são disponibilizadas;
- A mensuração do estado de saúde da população;

A linguagem JAVA é atualmente a linguagem mais utilizada no mundo, possui uma comunidade grande e que vem crescendo a cada dia, por ser *open source* e multiplataforma, pontos que chamam muita atenção. Por ser Orientada a Objetos (DEITEL, 2013) permite uma maior facilidade no desenvolvimento e na manutenção de um sistema modular, flexível e escalável, aproveitando-se ainda de conceitos, tais como: hereditariedade, polimorfismo, encapsulamento e outros. Possui uma API valiosa, para I/O, *networking*, utilitários, *xmlparsing*, conexão com banco de dados, ferramentas poderosas para desenvolvimento como Eclipse, Netbeans e IntelliJ, com uma gama de facilidades e ajudas para o desenvolvedor, como *debugging*, *plugins*, auxílios no desenvolvimento do código e os famosos *autocompletes*. Vale ressaltar que o JAVA possui uma grande coleção de bibliotecas *open source*, e *frameworks* como Spring, Struts e Maven, que asseguram que o desenvolvimento seguem as melhores práticas, outro ponto forte que vale ser mencionado é a comunidade vasta e proativa, suporte e uma excelente documentação com exemplos utilizando uma linguagem de fácil entendimento. Tais características foram importantes na escolha dessa linguagem no projeto desenvolvido.

Na maioria dos sistemas operacionais modernos, um corpo formado por código reusável é organizado e disponibilizado para simplificar o trabalho do programador, este código geralmente encontra-se em forma de bibliotecas.

Um Projeto JAVA possui várias formas para realizar leitura e escrita de um arquivo “.DBF”, salientando-se a biblioteca “javadb-1.7.1” fornecida e mantida por Fernández (2017) e outros desenvolvedores da comunidade. A biblioteca possui uma classe DBFReader, que possui vários métodos para leitura de um arquivo “.DBF”, como o “*getField*” para recuperar os cabeçalhos do arquivo “.DBF” e o “*nextRecord*” para recuperar cada linha de dados do arquivo “.DBF”.

A maior dificuldade da leitura de arquivos DBC se dá pelo motivo de ser um formato desenvolvido internamente pelo DATASUS com pouca informação disponível ao seu respeito. O formato de arquivo DBC é basicamente um arquivo DBF compactado usando o algoritmo de "compressão" da Biblioteca de Compressão de Dados (DCL) do PKWare. Existem no mercado, alguns tipos de *software* que utilizam a extensão DBC, como por exemplo, o *database file* do Microsoft FoxPro, porém estes arquivos não são compatíveis com o formato do DATASUS. No entanto, o estudo de um projeto *open source* chamado *blast-dbf*, um programa de linha de comando escrito na linguagem C que tem a funcionalidade de converter (descompactar) arquivos DBC para DBF, viabilizou o entendimento do mecanismo de compressão empregado nos arquivos DBC.

A linguagem JAVA oferece suporte nativo para a importação de funções e bibliotecas escritas em outras linguagens, como o C, C++ e FORTRAN. Aproveitando-se desta possibilidade, o código de descompressão de arquivos DBC escrito em C foi transformado em uma *dynamic link library* (.DLL) para permitir o seu acesso pelo JNA, uma biblioteca que abstrai chamadas de métodos e os tipos de dados de uma linguagem para outra. Com ele, no final das contas, nem parece que você está chamando linguagem nativa, pois ele facilita muito a integração.

Posteriormente foi criada uma interface de fachada. Essa interface estende a interface *com.sun.jna.Library* e contém a declaração de todos os métodos que nossa aplicação irá acessar da biblioteca nativa (.DLL), a função em JAVA, chamada “*dbc2dbf*”, será responsável por chamar o método nativo *blast-dbf* e em tempo execução, criar um arquivo DBF a partir de qualquer arquivo DBC, podendo então ser lido pela classe DBFReader da biblioteca *javadb-1.7.1*, podendo ler seus cabeçalhos e dados.

Com os dados agora mais claramente lidos e interpretados, utilizou-se uma função “*dbWrite*” em JAVA responsável por criar e inserir em um banco de dados local, para permitir consultas/análises em linguagem SQL ou criação de outros sistemas derivados.

Este trabalho está estruturado da forma como segue. No Capítulo 2 é apresentada a fundamentação teórica. O Capítulo 3 apresenta a desenvolvimento do trabalho. O capítulo 4, 5 e 6 explicam e detalham o desenvolvimento da ferramenta explanando sua arquitetura e todos os passos necessários para utilização da ferramenta. O capítulo 7 mostra a aplicação de um questionário sobre a usabilidade da ferramenta. O capítulo 8 encerra com a conclusão do trabalho.

2. FUNDAMENTAÇÃO TEÓRICA

As ações do programa de Governo Eletrônico (eGOV) priorizam o uso das tecnologias da informação e comunicação (TICs) para democratizar o acesso à informação, visando ampliar o debate e a participação popular na construção das políticas públicas, como também aprimorar a qualidade dos serviços e informações públicas prestadas. (GOVERNO ELETRÔNICO, 2018),

A política de eGOV brasileiro segue um conjunto de diretrizes baseado em três ideias fundamentais: participação cidadã; melhoria do gerenciamento interno do Estado e integração com parceiros e fornecedores.

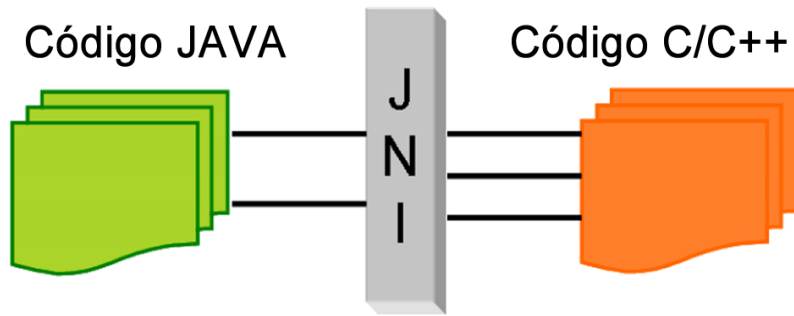
Transformar a relação do governo com a sociedade e promover interatividade com cidadãos, empresas e órgãos governamentais melhora o processo de democratização do país, dinamiza os serviços públicos e proporciona uma administração pública mais eficiente, já que, agora, a sociedade possui instrumentos para se manifestar junto às ações governamentais.

A disponibilização desses dados nem sempre estão em um arquivo que possa ser lido ou interpretável “humanamente”, necessitando assim de pré-processamentos para possibilitar a extração de informações. Existem situações em que será necessário você integrar duas aplicações, escritas em linguagens diferentes, por exemplo, uma escrita em C e outra em Java. Existem várias formas de se fazer isto, como escrever um *Web Service*, troca de arquivos, bancos de dados compartilhados etc.

Se existe apenas a necessidade de se usar um método de uma biblioteca nativa já existente, pode-se fazer o JAVA executar esse código diretamente, através de uma chamada nativa. Encontra-se em JAVA duas alternativas para o uso de um código nativo, o JNI (Java Native Interface) e o JNA (Java Native API).

2.1. Java Native Interface (JNI)

O *Java Native Interface* (JNI) é um padrão para escrever métodos nativos em Java, também conhecido como o *framework* padrão da ORACLE para acessar código em linguagem C (ou qualquer outra linguagem nativa como FORTRAN). (JAVA.COM, 2017)

Figura 1 - JNI (Java Native Interface)

Fonte: <<https://www.paulocollares.com.br/2013/01/um-simples-exemplo-de-jni-java-native-interface/>>

Esta abordagem é um pouco complexa, pois obriga que seu código nativo siga algumas regras de declaração de método que são trabalhosos e não triviais. Segue um exemplo de como somar dois números em C utilizando o JNI para chamá-lo nativamente JAVA.

Figura 2 - Assinatura do método desejado com modificador NATIVE

```

1
2 public class CalculadoraJNI {
3
4     public native int soma (int num1, int num2);
5
6 }

```

Fonte: Elaborado pelo autor (2017)

Para que a JVM carregue o código escrito em C é necessário a utilização de método *loadlibrary* da classe System.

Figura 3 - Carregando biblioteca nativa para a JVM

```

1
2 public class CalculadoraJNI {
3
4     public native int soma (int num1, int num2);
5
6     static {
7         System.LoadLibrary("somador");
8     }
9 }

```

Fonte: Elaborado pelo autor (2017)

A classe responsável por testar a calculadora seria algo conforme a Figura 3.

Figura 4 - Classe teste da calculadora

```

1
2 public class TestaCalculadoraJNI {
3     public static void main (String[] args) {
4         CalculadoraJNI cal = new CalculadoraJNI();
5         int num1 = Integer.parseInt(args[0]);
6         int num2 = Integer.parseInt(args[1]);
7         int resultado = cal.soma(num1, num2);
8         System.out.println("O resultado é: "+ resultado);
9     }
10 }

```

Fonte: Elaborado pelo autor (2017)

Após a fase de compilação e execução do código seria verificado um erro, pois ainda não seria encontrada uma biblioteca chamada “somador”, carregada na Figura 2. Aqui começa algumas regras necessárias para o sucesso da operação: é necessário que o método nativo em C seja o mesmo nome declarado no JAVA com a palavra reservada *NATIVE*, além da utilização de tipos específicos do JNI para a conversão entre as linguagens. A ferramenta JAVAH é responsável por gerar a assinatura que cuida dos detalhes de conversão de tipos.

Figura 5 - Utilização da ferramenta JAVAH

```

javah CalculadoraJNI

```

Fonte: Elaborado pelo autor (2017)

Após a execução do comando da Figura 5 foram criados 2 arquivos com extensões “.c” e “.h”. O Arquivo “somadorJNI.h” possui a declaração de métodos e tipos similares, o arquivo “somadorJNI.c” é onde será colocado a implementação do método de soma em linguagem nativa.

Figura 6 – Arquivo SomadorJNI.h

```

1 /* DO NOT EDIT THIS FILE - it is machine generated */
2 #include <jni.h>
3 /* Header for class CalculadoraJNI */
4
5 #ifndef _Included_CalculadoraJNI
6 #define _Included_CalculadoraJNI
7 #ifdef __cplusplus
8 extern "C" {
9 #endif
10 /*
11  * Class:    CalculadoraJNI
12  * Method:   soma
13  * Signature: (II)I
14  */
15 JNIEXPORT jint JNICALL Java_CalculadoraJNI_soma
16 (JNIEnv *, jobject, jint, jint);
17
18 #ifdef __cplusplus
19 }
20 #endif
21 #endif

```

Fonte: Elaborado pelo autor (2017)

Figura 7 - Arquivo SomadorJNI.c

```

1#include <stdio.h>
2#include "CalculadoraJNI.h"
3
4
5
6int soma (int num1, int num2){
7    int resultado = num1 + num2;
8    return resultado;
9}
10
11JNIEXPORT jint JNICALL Java_Calculadora_soma(JNIEnv * env, jobject obj, jint num1, jint num2){
12
13    return soma(num1, num2);
14}

```

Fonte: Elaborado pelo autor (2017)

Ao executar a compilação: `gcc -o libsomador.so -shared -I/caminho/para/jdk/headers somadorJNI.c`, será criado o arquivo “libsomador.so” no mesmo diretório, basta executar o código java:

Figura 8 - Executando código nativo

```

L6
L7$ java TestaCalculadoraJNI 2 3
L8$ A soma é: 5
L9

```

Fonte: Elaborado pelo autor (2017)

2.2. Java Native Access (JNA)

A maior dificuldade do JNI se encontra na sua complexidade, exige do programador algum conhecimento de C/C++, além de ser algo realmente cansativo de se programar.

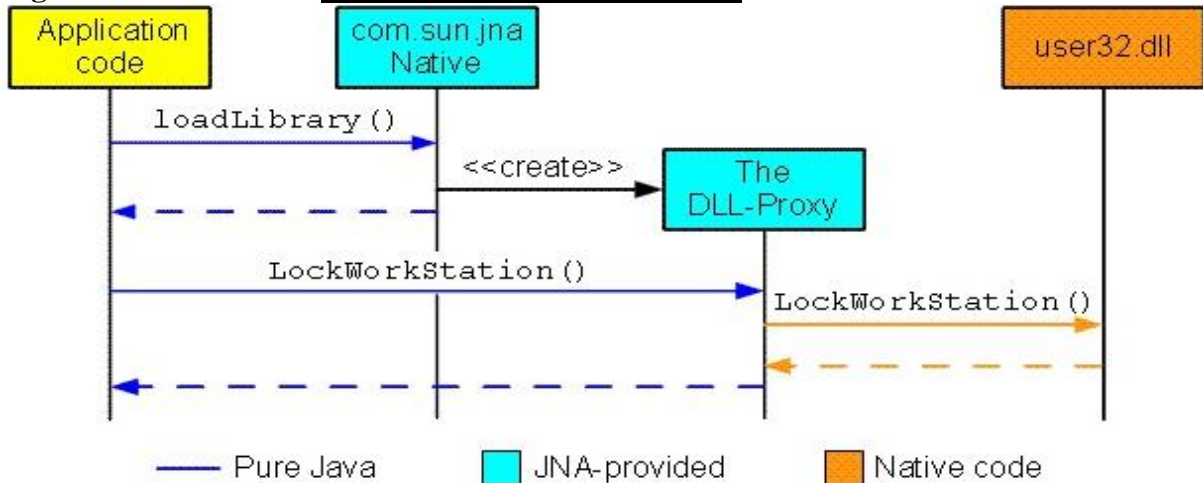
O JNI é uma biblioteca que abstrai chamadas nativas e os tipos de dados de uma linguagem para outra. Com ele no final das contas nem parece que você está chamando linguagem nativa, pois ele facilita muito a integração. (JNI, 2017)

De acordo com Devmedia (2017), a alternativa de usar o *framework* JNA (*Java Native Access*) se torna mais viável, pois abstrai as questões mais complexas do JNI tornando a integração entre código nativo e JAVA muito mais leve e produtiva.

O JNA fornece fácil acesso a bibliotecas compartilhadas nativas (DLLs) sem escrever nada além de código JAVA — não é necessário JNI ou código nativo, além de permitir a invocação direta de funções nativas usando a chamada do método JAVA natural. A maioria das chamadas não exige manipulação ou configuração especial, não é necessário nenhum código gerado.

O termo “DLL” não implica um formato específico e, portanto, a ferramenta que se deve usar em JAVA para chamar as funções depende do tipo de DLL que é. Poderia ser uma biblioteca .NET que contenha código gerenciado ou poderia ser uma DLL nativa com código não alterado (ambos dizem respeito à plataforma Windows) (Viana, 2016). Um código escrito em C por exemplo, pode ser compilado e logo após gerado uma DLL.

Figura 9 - JNA. Fonte: <https://www.devmedia.com.br/>



Fonte: Elaborado pelo autor (2017)

A criação automática do *DLL Proxy*, abstrai bastante o trabalho e comunicação entre o JAVA e a linguagem nativa.

No mercado encontram-se muitos projetos que utilizam o *framework* JNA, alguns destes descritos na Tabela 1.

Tabela 1 - Projetos que utilizam JNA

PROJETO	BREVE DESCRIÇÃO
Apache Cassandra	Armazenamento de dados NoSQL em grande escala.
IntelliJ	Ambiente de desenvolvimento
NetBeans	Ambiente de desenvolvimento
OpenVR	Acesso ao hardware

Fonte: <<https://github.com/java-native-access/jna>>

3. TRABALHOS CORRELATOS

Na literatura encontram-se trabalhos que se propõem a apresentar algumas formas de extração de dados gerais do DATASUS ou específicos para um determinado objetivo. São desenvolvidos estudos sobre as possíveis formas de captura, extração e interpretação das informações.

O trabalho desenvolvido por Adler (2016) foi a porta de entrada para o desenvolvimento de projetos que trabalham com a temática de leitura e interpretação de dados DBC. Arquivos em formato DBC são arquivos DBF comprimidos e criptografados em algoritmos privados (PKWare), por exemplo os arquivos de dados DATASUS. Seu trabalho contribuiu com o desenvolvimento de um descompactador de arquivos DBC que são condensados pela Biblioteca de Compressão PKWare.

Petruzalek (2016) propõe um desenvolvimento de um pacote para linguagem R com a finalidade de ler arquivos de dados do DATASUS em formato DBC, se baseando no trabalho de Adler (2016). Em sua metodologia a autora faz uma análise do formato do arquivo DBC e desenvolveu um *software* para a conversão deste formato para DBF e encapsulou esse código em um pacote para distribuição. O pacote chamado de “*read.dbc*” simplificou o processo de importação de dados do DATASUS na linguagem R, permitindo a leitura direta do arquivo DBC sem necessidade de conversão prévia.

4. BIBLIOTECA DATASUSDBCTODBF.JAR

Conforme exposto nos capítulos anteriores, a forma como o DATASUS disponibiliza os dados possui uma complexidade e um funcionamento direcionado apenas para a plataforma Windows. Neste caso, o acesso aos dados fica dependente da utilização de programas como TabWin, TabNet ou até mesmo o dbc2dbf.exe (disponível apenas em ambiente Windows) para processá-los e convertê-los para o formato DBF. Após essa conversão, ainda há a necessidade de programas terceiros para explorar tais arquivos, com o intuito de filtrar e extrair informações pertinentes que ajudariam em algum interesse específico.

Neste contexto, este trabalho tem por objetivo apresentar a criação de uma ferramenta open-source, multiplataforma e descomplicada. Esta será utilizada para extração e viabilização das informações dos dados DATASUS, visando atender todas as informações que sejam necessárias para o interesse de pessoas ligadas a pesquisa e/ou análise de dados.

Em busca realizada nos registros de programa de computador no site do INPI e no Google com as palavras chaves (DATASUS, DBC, Visualizador de dados da saúde e Pré-processamento), não foram encontrados cadastros de *softwares* semelhantes, tornando singular a ferramenta desenvolvida neste trabalho.

O DATASUS possui uma base de dados com as informações do cenário da saúde do povo brasileiro, as quais podem servir para subsidiar análises objetivas da situação sanitária, tomadas de decisão baseadas em evidências e elaboração de programas de ações de saúde. Dados de morbidade, incapacidade, acesso a serviços, qualidade da atenção, condições de vida e fatores ambientais passaram a ser métricas utilizadas na construção de Indicadores de Saúde, que se traduzem em informação relevante para a quantificação e a avaliação das informações em saúde. A Tabela 2 apresenta uma classificação dos dados que são disponibilizados e uma breve descrição destes.

Tabela 2 - Dados disponíveis no DATASUS.

BASE	DESCRIÇÃO
SIHSUS	Arquivos dissemináveis para tabulação do Sistema de Informações Hospitalares do SUS
SIASUS	Arquivos dissemináveis para tabulação do Sistema de Informações Ambulatoriais do SUS
SIM	Arquivos dissemináveis para tabulação do Sistema de Informações de Mortalidade
CIH	Arquivos dissemináveis para tabulação do Sistema de Comunicação de Informação Hospitalar
CIHA	Arquivos dissemináveis para tabulação do Sistema de Comunicação de Informação Hospitalar e Ambulatorial

SINASC	Arquivos dissemináveis para tabulação do Sistema de informação de Nascidos Vivos
SISPRENATAL	Arquivos dissemináveis para tabulação do Sistema de Monitoramento e Avaliação do Pré-Natal, Parto, Puerpério e Criança
CNES	Arquivos dissemináveis para tabulação do Cadastro Nacional de Estabelecimentos de Saúde
Base Territorial	Arquivos com a base territorial (municípios, regiões de saúde, territórios da cidadania etc.) em uso para a Disseminação de Informações de Saúde, assim como mapas e arquivos de conversão para uso pelo TabWin e TabNet.
Base Populacional	Arquivos com a distribuição da população brasileira segundo censos demográficos, contagens populacionais e estimativas, desde 1980, por município, sexo e idade, e segundo as estimativas realizadas para o TCU, desde 1992, por município.

Fonte: Elaborado pelo autor (2017)

Cada arquivo de dados possui um endereço FTP em que pode ser realizado seu *download*. Como exemplo, foi utilizado o CNES encontrado na URL: ftp://ftp.datasus.gov.br/dissemin/publicos/CNES/200508_/Dados/ST/, conforme a Figura 10.

Figura 10 - Repositório de Dados da CNES

Nome	Tamanho	Data da modificação
STAC0508.dbc	18.1 kB	05/06/2014 07:30:00
STAC0509.dbc	18.3 kB	05/06/2014 07:30:00
STAC0510.dbc	17.3 kB	05/06/2014 07:30:00
STAC0511.dbc	17.3 kB	05/06/2014 07:30:00
STAC0512.dbc	17.4 kB	05/06/2014 07:30:00
STAC0601.dbc	17.4 kB	05/06/2014 07:30:00
STAC0602.dbc	17.6 kB	05/06/2014 07:30:00
STAC0603.dbc	17.6 kB	05/06/2014 07:30:00
STAC0604.dbc	17.7 kB	05/06/2014 07:30:00
STAC0605.dbc	17.7 kB	05/06/2014 07:30:00
STAC0606.dbc	18.2 kB	05/06/2014 07:30:00
STAC0607.dbc	18.9 kB	05/06/2014 07:30:00

Fonte: Elaborado pelo autor (2017)

A nomenclatura de cada arquivo do repositório possui uma sintaxe, uma forma de identificação de algumas informações através da sua descrição. Afim de facilitar a explicação, adotou-se como exemplo o arquivo exposto na Figura 11, com a divisão em blocos.

Figura 11 - Nomenclatura arquivo CNES

STAC0508.dbc

1º 2º 3º 4º

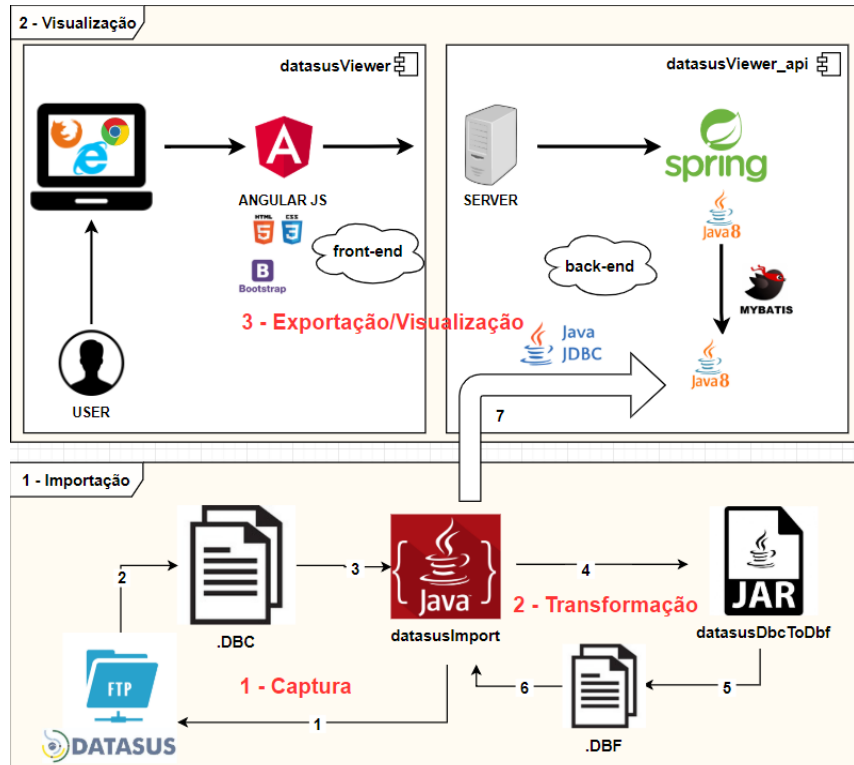
Fonte: Elaborado pelo autor (2017)

- **1º Bloco:** subgrupo de arquivo
- **2º Bloco:** UF do Estado
- **3º Bloco:** mês da consulta
- **4º Bloco:** ano da consulta

4.1. Arquitetura da Ferramenta

A Figura 12 apresenta a arquitetura da biblioteca proposta neste trabalho. Analisando a forma de funcionamento, pode-se dividir a arquitetura em 3 fases: Captura; Transformação; e Visualização de dados.

Figura 12 - Arquitetura ferramenta de pré-processamento e visualização de dados do DATASUS utilizando a biblioteca datasusDbcToDbf.jar



Fonte: Elaborado pelo autor (2017)

A seguir são detalhadas cada uma das fases da arquitetura.

4.1.1. Captura

Conforme apresentado na seção 3.1, o DATASUS disponibiliza as informações em formato .DBC por meio de um *link* FTP. Nessa fase, a biblioteca criada está apta para realizar o download dessas informações. A Figura 13 mostra o trecho de código responsável por essa etapa, encontrado do projeto *datasusImport*.

Figura 13 – Código da etapa de captura dos dados

```

String STATE = mainFrame.getComboBoxEstado().getSelectedItem().toString();
String MONTH = mainFrame.getComboBoxMes().getSelectedItem().toString();
EnumDomainFileDatasus DOMAIN = EnumDomainFileDatasus.CNES;
String YEAR = mainFrame.getComboBoxAno().getSelectedItem().toString().substring(2,4); //2018;
String TYPE_FILE = mainFrame.getComboBoxTipo().getSelectedItem().toString();
ConnectionDbUtils db = null;
Connection con = null;

try {
    String pathDBF;
    String pathDBC;
    File fileDBF = null;
    DBFReaderUtils dbfReader = null;
    String filename = FileUtils.createFileName(STATE, YEAR, MONTH, TYPE_FILE);
    String urlDownload = FileUtils.createUrlDownload(filename, TYPE_FILE, DOMAIN.getUrl());
    pathDBC = FileUtils.createPathFileDBC(filename, DOMAIN.getInitial(), TYPE_FILE);
    File fileDBC = new File(pathDBC);

    mainFrame.getConsoleTextArea().append("##### REALIZANDO DOWNLOAD: " + urlDownload + "\n");
    NetUtils.downloadUsingStream(urlDownload, fileDBC);
}

```

Fonte: Elaborado pelo autor (2017)

Com o auxílio da classe URL do pacote *java.net.URL*, a biblioteca acessa o link FTP, fazendo o *download* do arquivo DBC e salva em uma pasta escolhida pelo usuário.

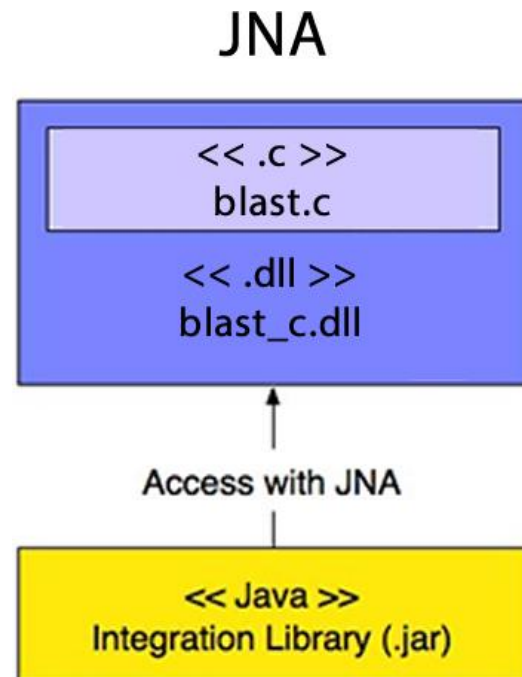
4.1.2. Transformação

Essa fase é a parte principal de operação da biblioteca *dbcToDbf-1.1.jar*. Após os arquivos DBC serem baixados e salvos em uma pasta local, iniciando assim sua descryptografia e posteriormente sua transformação.

A extensão “.DBC” é uma compreensão dos arquivos “.DBF”. Esta compressão é realizada por meio de um algoritmo de compressão privado. No caso do DATASUS, é utilizado a biblioteca de compressão *PKWare Compression*. Neste sentido, faz-se uso do código “blast.c” (ANEXO I) e o “blast.h” (ANEXO II) desenvolvido na linguagem C por Adler (2016), a fim de descryptografar os arquivos DBC.

Para que fosse possível a utilização desta biblioteca escrita em C na estrutura da ferramenta desenvolvida em Java, utilizou-se o JNA, (DOUBROVKINE, 2017), o qual permite um fácil acesso a bibliotecas nativas compartilhada. A Figura 14 destaca esta integração. Mais detalhes sobre o funcionamento do JNA foram expostos no Capítulo “Referencial Teórico”.

Figura 14 - JNA aplicado na biblioteca *dbcToDbf-1.1.jar*



Fonte: Elaborado pelo autor (2017)

Para viabilizar o uso do código “blast”, primeiramente se faz necessário a criação de uma DLL. Para criação da *blast_c.dll* foi utilizado o TDM-GCC 4.9.2 64-bit Release, conforme exposto na Figura 15.

Figura 15 - Comandos criação *blast_c.dll*

```
$ gcc.exe -c blast.c -o blast.o -I "C:/Program Files (x86)/Dev-Cpp/MingGW64/include"
-I "C:/Program Files (x86)/Dev-Cpp/MingGW64/x86_64-w64-mingw32/include"
-I "C:/Program Files (x86)/Dev-Cpp/MingGW64/lib/gcc/x86_64-w64-mingw32/4.9.2/include" -DBUILDING_DLL=1

$ gcc.exe -c dbc2dbf.c -o dbc2dbf.o -I "C:/Program Files (x86)/Dev-Cpp/MingGW64/include"
-I "C:/Program Files (x86)/Dev-Cpp/MingGW64/x86_64-w64-mingw32/include"
-I "C:/Program Files (x86)/Dev-Cpp/MingGW64/lib/gcc/x86_64-w64-mingw32/4.9.2/include" -DBUILDING_DLL=1

$ gcc.exe -shared blast.o dbc2dbf.o -o dll.dll
-L "C:/Program Files (x86)/Dev-Cpp/MingGW64/include"
-L "C:/Program Files (x86)/Dev-Cpp/MingGW64/x86_64-w64-mingw32/include"
-static-libgcc -Wl, --output-def,libdll.def, --out-implib, libdll.a,-add-stdcall-alias
```

Fonte: Elaborado pelo autor (2017)

Após a criação da DLL, a biblioteca passa a possuir uma classe principal chamada *CBlastLibrary.java*. Em sua construção, esta realiza o *loadLibrary("blast_c.dll")*, conforme explicado sobre a arquitetura JNA na Figura 9. Com isso, esta classe se beneficia da abstração de toda complexidade da chamada de códigos nativos no Java. Devido esta facilidade pode-se acessar o código nativo do trabalho de Adler (2016) para descomprimir o DBC e convertê-lo em DBF, conforme exposto no trecho de código da Figura 16.

Figura 16 - Método Java chamando código Nativo em C de conversão DBC em DBF

```
public void dbc2dbf(String in, String out) {
    this.cLibraryInstance.dbc2dbf(in, out);
}
```

Fonte: Elaborado pelo autor (2017)

Após realizar os passos anteriores, teremos a disponibilização do arquivo .DBF no local escolhido.

4.1.3. Visualização de dados

Após finalizado a etapa de transformação, um arquivo “resultado.dbf” é disponibilizado no local indicado pelo usuário. Nesta etapa, a ferramenta disponibiliza um método responsável por ler o arquivo “.DBF”, posteriormente criar uma tabela em um banco de dados com os cabeçalhos encontrados na primeira linha do arquivo, e assim, inserir na tabela as informações encontradas nas linhas posteriores.

As informações agora gravadas em um banco de dados ampliam a possibilidade de desenvolvimento de um *software* capaz de permitir a visualização de qualquer informação que seja interessante para subsidiar análises objetivas da situação sanitária, tomadas de decisão baseadas em evidências e elaboração de programas de ações de saúde. Por exemplo: quais os estados que oferecem mais serviços em determinada área médica, e quais os que oferecem menos? Com o passar dos anos a disponibilidade desses serviços vem aumentando? Essas são apenas umas das várias informações que podem ser extraídas com a utilização da ferramenta desenvolvida para que o governo possa tomar alguma decisão que beneficie a população.

Como mostrado na Tabela 2, existem vários outros arquivos que possuem informações que vão desde mortalidade infantil, Monitoramento e Avaliação do Pré-Natal, Parto, Puerpério e Criança, até informações de estabelecimentos de saúde.

4.2. Exemplo de utilização da biblioteca *Dbctodbf-1.1.Jar*

Como exemplo do uso da ferramenta, foi criado um pequeno código fonte (APÊNDICE I) que terá a função de importar a biblioteca criada, para buscar informações do DATASUS e guardar em um sistema gerenciador de banco de dados (PostgreSQL 9.1) local.

Como fonte de dados, foi utilizado o arquivo STMA1017.dbc, o qual possui público alvo os estabelecimentos Públicos de Saúde, Rede Complementar e Prestadores do SUS, tanto pessoas físicas quanto jurídicas do Estado do Maranhão no mês outubro de 2017.

O trecho “import win.CBlastLibrary” é responsável pela importação da ferramenta desenvolvida nesse trabalho. O código inicia com a identificação de qual arquivo deseja-se fazer o *download* no repositório DATASUS, o arquivo então é baixado e salvo em um diretório temporário. Posteriormente esse .DBC é utilizado na função “dbc2dbf” da ferramenta desenvolvida e convertido para um “.DBF”, salvo em um diretório também escolhido pelo usuário, e finalmente uma função irá persistir os dados contidos no arquivo em um banco de dados.

Após a transformação do arquivo “.DBC” pela biblioteca desenvolvida, o método responsável por gravar no banco de dados o arquivo DBF foi executado. O resultado desta inserção pode ser observado na Figura 17.

Figura 17 - Banco de dados com informações DATASUS

	cnes	codufmun	cod_cep	cpf_cnpj	pf_pj	niv_dep	cnpj_man	cod_ir	r
1	2360489	210005	65930000	00000000000000	3	3	07727589000173		10
2	2360497	210005	65930000	00000000000000	3	3	07611890000117		10
3	2360500	210005	65930000	00000000000000	3	3	07000268000172		10
4	2462923	210005	65930000	00000000000000	3	3	07000268000172		10
5	2462931	210005	65930000	00000000000000	3	3	07000268000172		10
6	2462958	210005	65930000	00000000000000	3	3	07000268000172		10
7	2462966	210005	65930000	00000000000000	3	3	07000268000172		10
8	2462990	210005	65930000	00000000000000	3	3	07000268000172		10
9	2463008	210005	65930000	00000000000000	3	3	07000268000172		10

Fonte: Elaborado pelo autor (2017)

No próximo capítulo, serão apresentados os resultados obtidos até o momento com a ferramenta desenvolvida.

5. ANÁLISE DE DESEMPENHO DA BIBLIOTECA

A fim de avaliar o desempenho da ferramenta criada, foi realizada a comparação com a disponibilizada por Petruzalek (2016), o qual foi apresentado na seção “2.3 - Trabalhos Correlatos”. Antes de apresentar os resultados da comparação é importante dissertar sobre as linguagens R, utilizada por esse autor e a JAVA, adotada na solução proposta.

Faz-se necessário lembrarmos uma característica importante de cada linguagem: a linguagem R é interpretativa, segundo Hornik (s.d.), ou seja, consiste na tradução dos comandos da linguagem intermediária para linguagem de máquina, enquanto a JAVA é considerada uma linguagem híbrida (compilada e interpretada pela JVM), ou seja, a JVM compila gerando bytecodes e a JVM interpreta os bytecodes de acordo com o sistema operacional, aplicando otimizações que resultam em um melhor aproveitamento dos múltiplos núcleos do ambiente. Zapalowski (2012) em seu trabalho fez uma análise quantitativa e comparativa de linguagens de programação entre as linguagens de programação interpretadas e as compiladas, onde após vários testes executados nas mesmas condições, com os mesmos problemas (laços encadeados, Função Ackermann, Cliente-Servidor), conclui-se que o desempenho das linguagens que usam compiladores são melhores do que as linguagens que usam interpretadores, e mesmo o JAVA sendo um meio termo, teve um desempenho maior que todas as linguagens interpretativas, levando-se em consideração métricas como linhas de código, tempo de execução e eficiência relativa.

Em seu trabalho Cribari-Neto (2013), lista algumas desvantagens que fazem com que a linguagem R tenha um desempenho inferior a linguagens compiladas, dentre elas:

- R é uma linguagem interpretada. Linguagens interpretadas tendem a ser mais lentas que linguagens compiladas;
- R pode ser lenta para trabalhar com problemas computacionalmente intensivos, como por exemplo, simulações de Monte Carlo, *bootstrap* duplo.
- R não foi pensado inicialmente para dar suporte à computação *multicore* apesar de hoje existirem pacotes para paralelização de processos;
- R é geralmente lenta em funções recursivas;
- Chamar um código C/C++ pode não ser uma tarefa simples para a maioria dos usuários da linguagem, diferentemente de como foi mostrado em JAVA, uma forma simples.

O cenário usado para a comparação em questão utilizou três arquivos “.dbc” obtidos na fonte de dados públicas do DATASUS de tamanhos 194 kb, 938 kb e 1.5 mb, respectivamente. Os testes foram executados sobre a mesma configuração de *hardware*: Processador Intel Core i7-6700HQ 2.60 Ghz, Memória RAM 16GB e cache de 5mb, a fim de que o resultado esperado fosse o mais coerente possível.

A Figura 17 e Figura 18 mostram os trechos de código criados nas linguagens R e JAVA, respectivamente, para a leitura dos arquivos “.dbc” e geração de um arquivo “.dbf” resultante da transformação.

Figura 17 - Trecho de Código em R

```
start.time <- Sys.time();
caminhoArquivo= paste(getwd(), "/DADOS/", dominio, "/", tipoArquivo, "/", nomeArquivo, sep = "")
j <- read.dbc(caminhoArquivo);
print(Sys.time() - start.time);
```

Fonte: Elaborado pelo autor (2017)

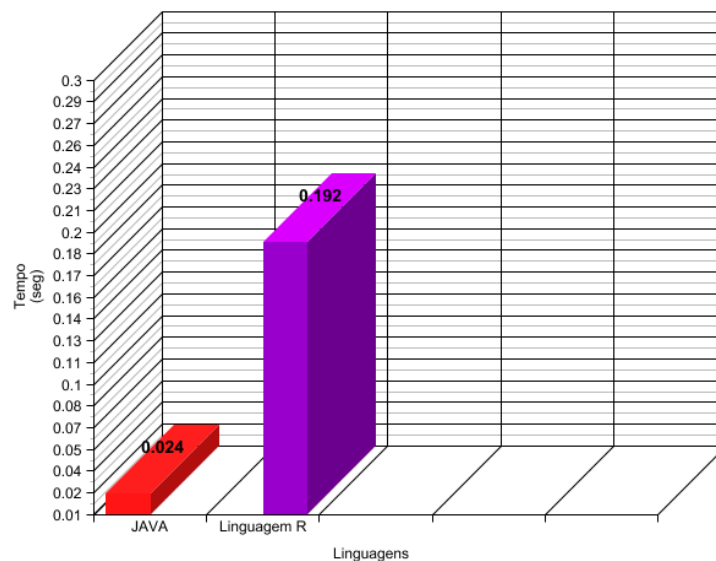
Figura 18 - Trecho de código em JAVA - dbcToDbf-1.1.jar

```
System.out.println("##### CRIANDO ARQUIVO .DBF EM PASTA TEMPORÁRIA: "+pathDBF);
CBlastLibrary cLib = new CBlastLibrary();
long tempoInicial = System.currentTimeMillis();
cLib.dbc2dbf(fileDBC.getAbsolutePath(), pathDBF);
cont=(System.currentTimeMillis() - tempoInicial)/1000.00;
System.out.println("o metodo executou em " + cont);
```

Fonte: Elaborado pelo autor (2017)

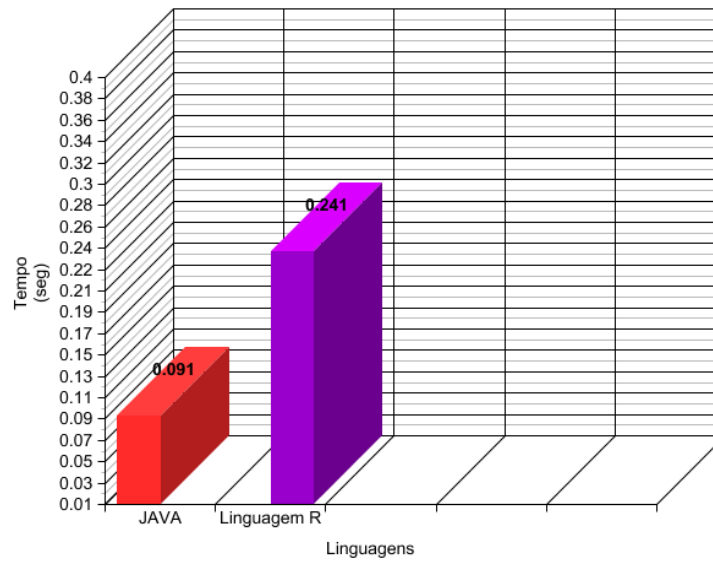
Para garantir a maior confiabilidade dos resultados foram efetuados 20 vezes o mesmo teste em cada arquivo A B e C, sendo a média de tempo apresentada na Tabela 3.

Figura 19 – Comparação de execução usando Arquivo DBC de 194 kb



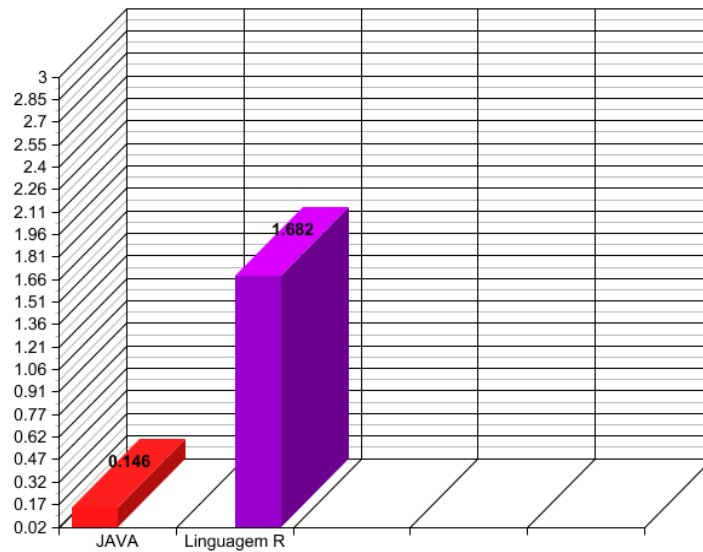
Fonte: Elaborado pelo autor (2017)

Figura 20 - Comparação de execução usando Arquivo DBC de 938 kb



Fonte: Elaborado pelo autor (2017)

Figura 21 - Comparação de execução usando Arquivo DBC de 1.5 Mb



Fonte: Elaborado pelo autor (2017)

Observou-se durante os experimentos que a biblioteca JAVA mostrou-se mais rápida na conversão do arquivo “.DBC” em “.DBF” do que a mesma desenvolvida em Linguagem R, a Tabela 3 mostra em detalhes a comparação.

Tabela 3 - Resultados comparativos da biblioteca JAVA e R

ARQUIVO	QUANTIDADE DE TESTES	RESULTADO
Arquivo 194kb	20	JAVA 8 vezes mais rápido que a Linguagem R
Arquivo 938kb	20	JAVA 2.8 vezes mais rápido que a Linguagem R
Arquivo 1.5mb	20	JAVA 11.5 vezes mais rápido que a linguagem R

Fonte: Elaborado pelo autor (2017)

Apesar de perceber no experimento com o arquivo de 938kb, tamanho do menor arquivo encontrado, que não houve um aumento de desempenho tão alto se comparado com os arquivos de 194kb e 1.5mb, ficando na casa de 2.8 vezes mais rápido, em nenhum dos testes percebeu-se um desempenho melhor utilizando a linguagem R.

Conforme mostrado na Figura 10, existe um arsenal de dados públicos que podem ser baixados em formato “.dbc”, mensal, para cada Estado da federação brasileira, a variação de tamanho dos arquivos tem em média 100kb a 5Mb de tamanho, em uma rápida pesquisa realizada no site do DATASUS. Logo conclui-se que biblioteca desenvolvida em JAVA se encaixa bem no contexto de conversão do arquivo “.DBC” em arquivo “.DBF”, de uma forma mais rápida que na linguagem R.

6. FERRAMENTA DE EXPORTAÇÃO E VISUALIZAÇÃO DE DADOS DO DATASUS

Neste tópico será explanado de forma mais profunda a etapa de visualização/exportação resumidamente mostrada no tópico 4.1.3. O objetivo é exemplificar por meio da criação de um *software* a dimensão do que pode ser construído a partir da biblioteca *datasusDbcToDbf* desenvolvida neste trabalho, de acordo com o interesse de quem utiliza a biblioteca.

Para demonstração foi produzido um *software*, observando as melhores práticas conforme Jones (2009), que faz uso da biblioteca *datasusDbcToDbf* (apresentada nas seções anteriores), e será detalhado sua estrutura e arquitetura, até a exportação de informações relevantes para um determinado *stakeholder*.

6.1. Arquitetura do Sistema

O sistema foi desenvolvido utilizando a tecnologia *client-side*, em que o lado do cliente (navegador) apenas dispara requisições HTTP para uma API (*Application Programming Interface*), esta por sua vez retorna ou executa uma ação e o lado do cliente em posse dessas informações JSON, renderizam a visão da página no navegador para o cliente.

Existem 2 etapas envolvidas para o funcionamento do sistema:

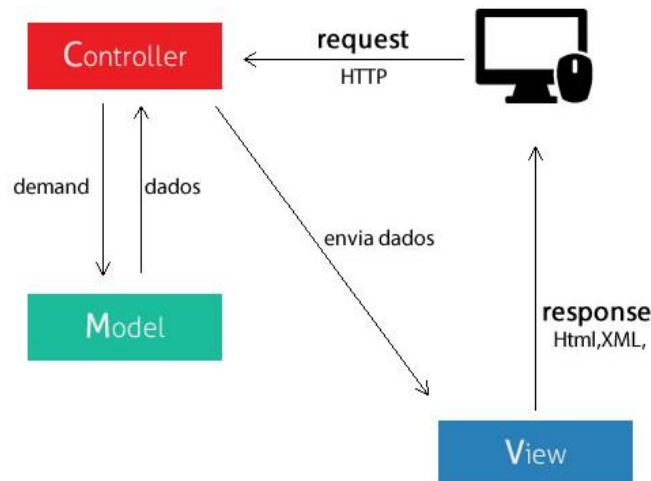
- A importação dos dados (*datasusImport*), onde todo o processo de *download* dos arquivos .DBC, sua descryptografia e inserção em um banco de dados relacional será realizada.
- A visualização dos dados (*datasusViewer* e *datasusViewer_api*), onde será realizada toda interação com o usuário, capturadas pelo *datasusViewer*, enviados para o *datasusViewer_api* e retornados para o *datasusViewer*, possibilitando assim: criação de gráficos dinâmicos e exportação de planilhas.

Os componentes do sistema, conhecidos como *front-end* e *back-end* foram desenvolvidos utilizando a arquitetura MVC (Modelo-Visão-Controle), em que a ideia central é reuso de código e separação de conceitos.

- Camada Modelo: é responsável pela leitura e escrita de dados, e também de suas validações.

- Camada Visão: a camada de interação com o usuário. Ela apenas faz a exibição dos dados, sendo ela por meio de um html, xml ou json.
- Camada Controle: O responsável por receber todas as requisições do usuário. Seus métodos chamados actions são responsáveis por uma página, controlando qual *model* usar e qual *view* será mostrado ao usuário.

Figura 22 - MVC

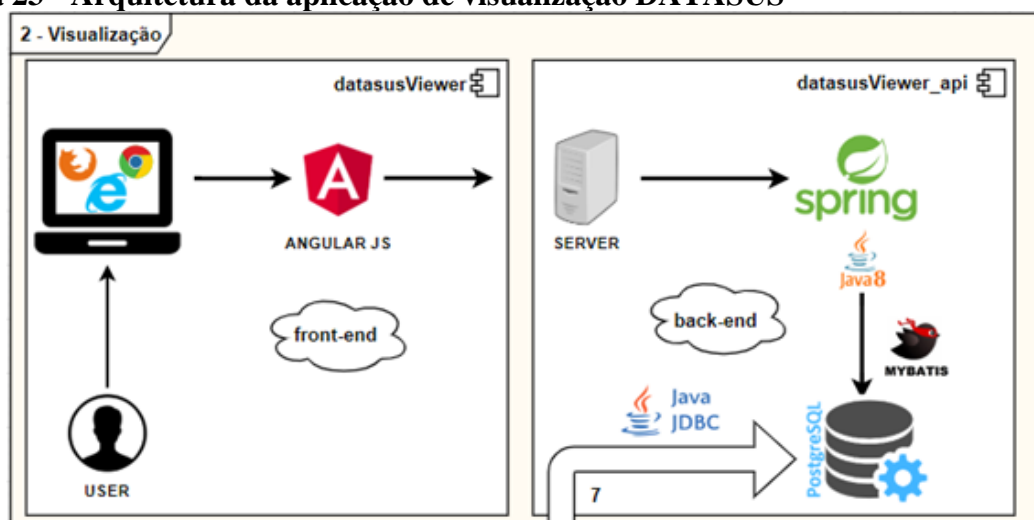


Fonte: <<https://tableless.com.br/mvc-afinal-e-o-que>>

Os *Requests* da aplicação serão realizados pelo *front-end*, que consumirá o retorno em formato json do *back-end*, para renderizar a visão para o cliente.

Na Figura 23 pode-se ter uma visão geral da arquitetura do sistema e como será atuado cada componente.

Figura 23 - Arquitetura da aplicação de visualização DATASUS



Fonte: Elaborado pelo autor (2017)

Neste sentido será detalhado as principais tecnologias escolhidas na arquitetura.

6.1.1. Angular JS

É um *framework* de *javascript* mantido pela Google, que torna as telas do sistema mais dinâmicas, fazendo a renderização parcial da página, fazendo com que seja menor o consumo do tráfego na rede e otimizando o carregamento. Também, possibilita que parte do processamento seja realizada na máquina do usuário logo poupando o servidor e o dedicado apenas para o processamento das rotinas de maior custo computacional e acesso ao banco de dados.

6.1.2. JAVA

É uma linguagem de programação multiplataforma compilada e interpretada por uma máquina virtual própria, com um vasto conjunto de bibliotecas de métodos e funções consolidados pela comunidade ativa da linguagem, possibilitando a realização de multitarefas, criação de um canal de comunicação via socket. Por esses motivos, sua utilização para integração dos componentes é ideal.

6.1.3. SPRING

É um *framework opensource* para a plataforma Java. No Spring, o *container* se encarrega de "instanciar" classes de uma aplicação Java e definir as dependências entre elas através de um arquivo de configuração em formato XML, inferências do *framework*, o que é chamado de *auto-wiring* ou ainda anotações nas classes, métodos e propriedades. Dessa forma o Spring permite o baixo acoplamento entre classes de uma aplicação orientada a objetos.

O Spring possui uma arquitetura baseada em interfaces e POJOs (*Plain Old Java Objects*), oferecendo aos POJOs características como mecanismos de segurança e controle de transações. Também facilita testes unitários e surge como uma alternativa à complexidade existente no uso de EJBs. Com Spring, pode-se ter um alto desempenho da aplicação.

Esse *framework* oferece diversos módulos que podem ser utilizados de acordo com as necessidades do projeto, como módulos voltados para desenvolvimento Web, persistência, acesso remoto e programação orientada a aspectos.

6.1.4. MyBatis

MyBatis é uma estrutura de persistência de primeira classe com suporte a SQL customizado, procedimentos armazenados e mapeamentos avançados. MyBatis elimina quase todo o código JDBC e configuração manual de parâmetros e recuperação de resultados. O MyBatis pode usar XML simples ou Anotações para primitivas de configuração e mapa, interfaces de Mapa e Java POJOs (Plain Old Java Objects) para registros de banco de dados.

6.1.5. Java JDBC

É um conjunto de classes e interfaces (API) escritas em Java que fazem o envio de instruções SQL para qualquer banco de dados relacional.

- Api de baixo nível e base para api's de alto nível;
- Amplia o que você pode fazer com Java;
- Possibilita o uso de bancos de dados já instalados.

6.2. Modelo de Dados

Para início do processo de modelagem é importante o entendimento de como a solução, deve realizar suas funções. De forma geral, a partir do que foi analisado de pontos negativos da ferramenta do DATASUS, para visualização das informações, que são muito limitadas tecnicamente e cujo código fonte não é aberto, impossibilitando assim que qualquer pessoa utilize, ou possa adaptar da forma que achar conveniente para seu uso, seja para informação ou para apoiar análises objetivas da situação sanitária, tomadas de decisão baseadas em evidências e elaboração de soluções de ações de saúde.

Foi observado a necessidade de desenvolver o sistema seguindo os requisitos abaixo:

- Aquisição de dados essenciais para o desenvolvimento e armazenamento em uma base de dados consolidada, a ser utilizada para realização agrupamentos e exportação dos dados processados;
- Disponibilizar interface capaz de realizar a parametrização de dados, a partir de uma base de dados previamente carregada, por meio de interface web;
- Gerar relatórios dos dados referentes aos resultados das simulações, bem como disponibilizar os dados em formatos comuns no mercado (CSV e PDF);

- Gerenciar o acesso à ferramenta, estabelecendo critérios de manipulação de suas funcionalidades.

A partir do levantamento das informações que sanariam as dificuldades relatadas sobre a captação/visualização dos dados do DATASUS, foi escolhido um cenário dentre vários outros que o DATASUS possui, para demonstração da utilização da ferramenta, desde captura dos dados,criptografia e importação em um banco de dados relacional até sua visualização. Foi escolhido para demonstração o domínio de dados CNES (Arquivos dissemináveis para tabulação do Cadastro Nacional de Estabelecimentos de Saúde), e o subgrupo do tipo ST (estabelecimentos) e PF (profissionais), com intuito de possibilitar a visualização do cenário de saúde brasileiro em relação a prestação de especialidades médicas fornecidas por estabelecimentos, agrupadas por estados, municípios, tipo de estabelecimento, período de captação dos dados.

6.3. Modelo de entidade e relacionamento

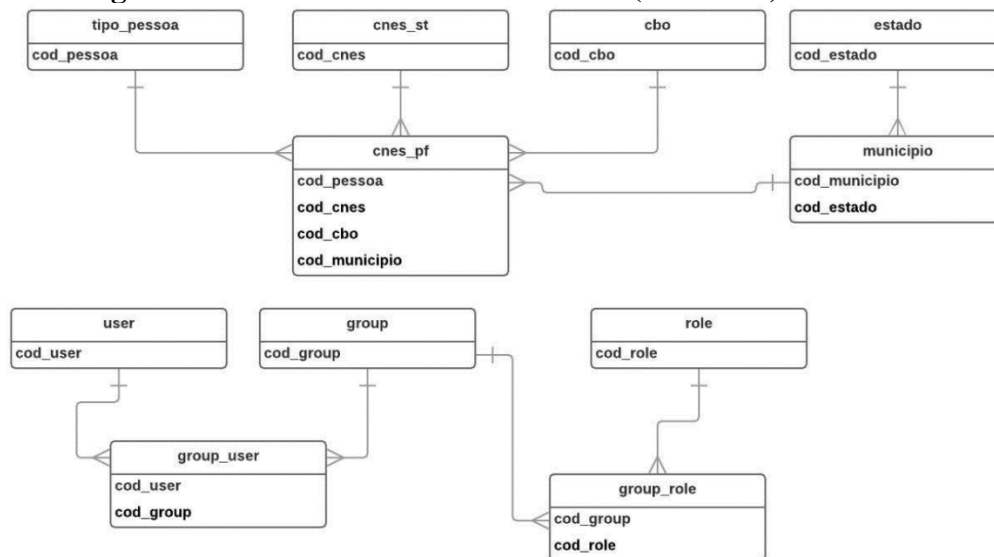
Modelar dados significa criar um modelo que explique as características de funcionamento e comportamento de um *software* a partir do qual ele será criado, facilitando seu entendimento e seu projeto. Permitem demonstrar como serão construídas as estruturas de dados que darão suporte aos processos de negócio, como esses dados estarão organizados e quais os relacionamentos que pretendemos estabelecer entre eles.

Para a construção do Modelo de Dados, foram empregadas as técnicas do Diagrama Entidade Relacionamento (DER), com objetivo de representar esquematicamente os dados em um nível de detalhamento específico para cada visão esperada.

O DER promove a construção de modelos de dados como ponto de partida para a implementação de sistemas informatizados, enfatizando o aspecto lógico das entidades do banco de dados (em forma de tabelas), as ligações entre as tabelas de banco de dados, as chaves primárias, os tipos de dados de cada atributo.

A Figura 24 ilustra o DER do sistema.

Figura 24 - Diagrama de Entidade e Relacionamento (resumido)



Fonte: Elaborado pelo autor (2017)

O dicionário de dados usado para detalhar o conteúdo de cada um dos campos utilizados nas tabelas é apresentado.

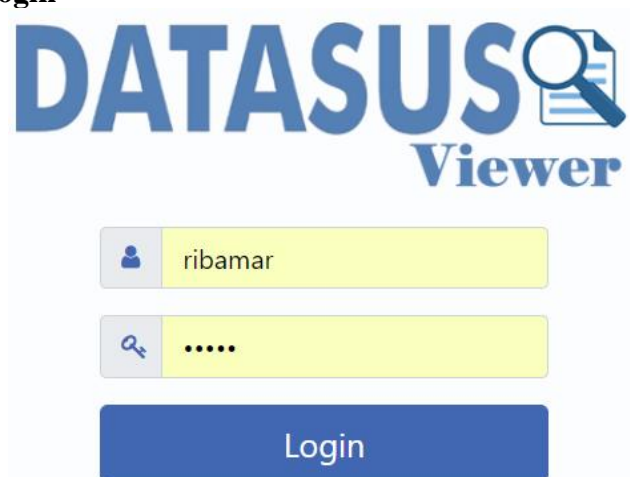
Tabela 4 - Dicionário de Dados do DER

Entidade / Conceito	Definição
tipo_pessoa	Entidade que define o tipo de pessoa: PF – pessoa física, PJ – pessoa jurídica
cnes_st	Entidade que possui as informações do DATASUS referentes aos estabelecimentos de saúde cadastrados na base nacional, tabela será criada e preenchida pelo datasisImport
lista_cbo	Entidade que possui a lista de classificações brasileira de ocupações
Estado	Entidade que possui pelos estados brasileiros
município	Entidade que possui os municípios brasileiros
cnes_pf	Entidade que possui as informações do DATASUS referentes aos profissionais de saúde cadastrados na base nacional, tabela será criada e preenchida pelo datasisImport
User	Entidade que possui os usuários do sistema
Group	Entidade que possui os grupos de permissões dos usuários
role	Entidade que possui as permissões que o sistema possui
role_group	Entidade que possui as permissões que um grupo de usuário possui
group_user	Entidade que possui os grupos de usuário que o sistema possui

Fonte: Elaborado pelo autor (2017)

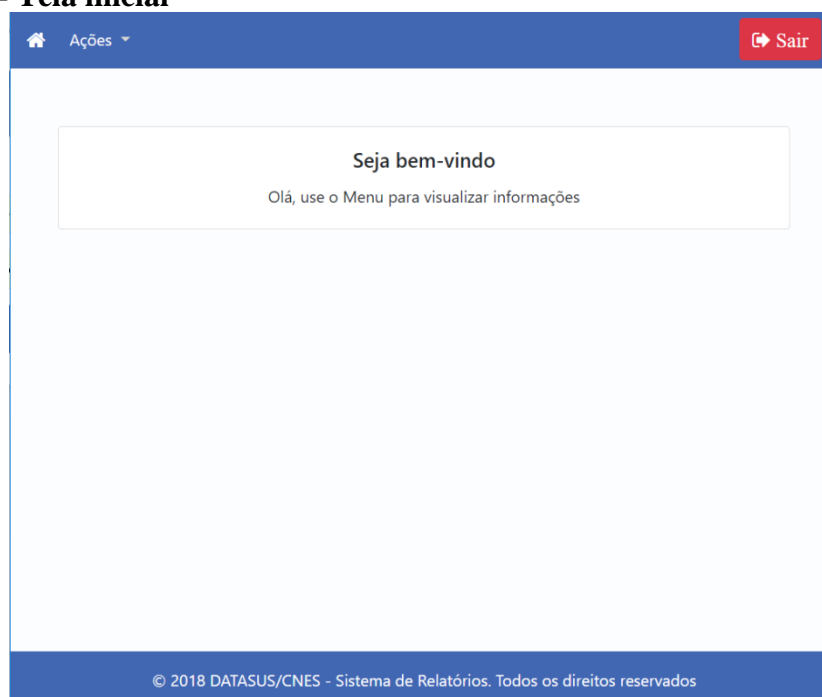
6.4. Interface gráfica

A aplicação de visualização/exportação de dados se inicia com uma tela de login para permitir acesso à navegação das páginas, conforme a Figura 25.

Figura 25 - Tela de login

Fonte: Elaborado pelo autor (2017)

Após concluído o processo de autenticação, ocorre um redirecionamento automático para a tela inicial (Figura 26). Nesta página pode ser encontrada uma barra de menu no topo, responsável pelos redirecionamentos internos na aplicação.

Figura 26 - Tela inicial

Fonte: Elaborado pelo autor (2017)

6.4.1. Captura e Transformação

Conforme explicado anteriormente, o processo de captura de dados do DATASUS consistem nas seguintes etapas: 1) receber o arquivo DBC ainda em seu estado “puro” (informações criptografadas e que não podem ser interpretadas humanamente) da base de

dados; 2) descritografia das informações utilizando o trabalho de Adler (2016) somada com a técnica de JNA da linguagem JAVA, resultando assim em um arquivo interpretável; 3) armazenamento dos dados em um banco de dados relacional, para que assim possa ser utilizado para o fim necessário.

Na Figura 27 pode ser observada a tela responsável pelas entradas que devem ser fornecidas para a captura e transformação das importações. Essas informações servirão para identificar o arquivo que deve ser baixado, cuja estrutura foi explicada na seção 4

Figura 11.

Figura 27 - Tela de importação/transformação dos dados

A imagem mostra a interface de usuário para a importação de dados no sistema Datasus Viewer. O cabeçalho da página é azul escuro com o texto 'Ações' e um ícone de casa. O conteúdo principal é branco e contém o título 'Importação' com o ícone de um banco de dados e o subtítulo 'Datasus Viewer'. Abaixo disso, há quatro campos de seleção: '-- Mês --', '-- Ano --', '-- Estado --' e '-- Arquivo --'. À direita dos campos, há dois botões: 'Limpar' (branco) e 'Importar' (verde). No rodapé, há uma barra azul com o texto: '© 2018 DATASUS/CNES - Sistema de Relatórios. Todos os direitos reservados'.

Fonte: Elaborado pelo autor (2017)

A seguir o detalhamento de cada campo:

- **Mês:** campo onde deve ser informado qual mês deseja-se obter a informações. Ex: '01' = JANEIRO, '02' = FEVEREIRO
- **Ano:** campo onde deve ser informado qual ano deseja-se obter a informações. Ex: '18' = 2018, '17' = 2017
- **Estado:** campo onde deve ser informado qual estado deseja-se obter a informações. Ex: 'MA' = Maranhão, 'PI' = PIAUÍ

- **Arquivo:** campo onde deve ser informado qual arquivo deseja-se obter a informações, no caso em questão está sendo trabalhado os arquivos do domínio CNES. Ex: 'PF' = Profissionais, 'ST' – Estabelecimentos.

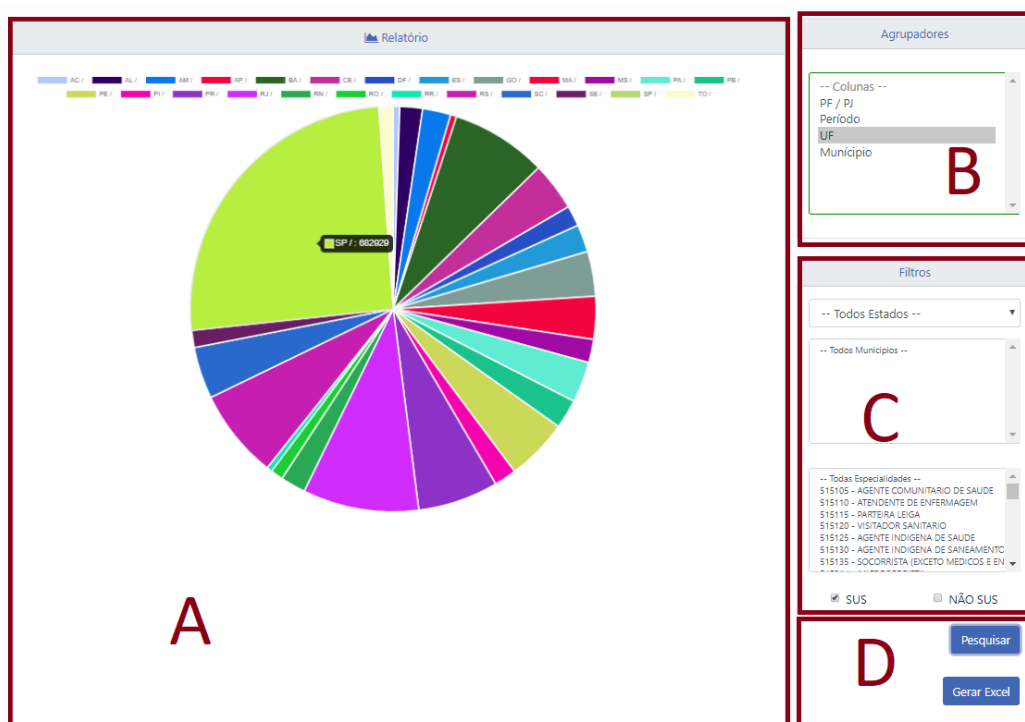
6.4.2. Visualização e Exportação de dados

Para o exemplo que será demonstrado por meio desta aplicação, foram utilizados somente os dados de 2018 referentes ao domínio que trata dos estabelecimentos de saúde e funcionários da saúde. A proposta desta aplicação visa demonstrar o panorama da distribuição de ofertas de especialidades médicas nos estados e municípios brasileiros.

A tela de relatórios é responsável pela visualização dessas informações em forma de gráfico ou exportação por meio de planilhas. A

Figura 28 apresenta um exemplo de relatório gerado pela ferramenta.

Figura 28 - Tela de Relatórios



Fonte: Elaborado pelo autor (2017)

A tela de relatórios se divide em quatro áreas:

- **Área A:** o resultado gráfico da consulta realizada, informando o quantitativo de agrupamento das colunas
- **Área B:** local onde são escolhidos os agrupadores da consulta quantitativa, que serão as legendas do gráfico

- **Área C:** filtros utilizados na consulta, que podem ser por estados, municípios, e especialidades médicas específicas que atendem pelo sistema SUS ou não.
- **Área D:** ações possíveis da aplicação. A ação “Pesquisar” montará o resultado em gráfico e ação “Gerar Excel” exporta as informações em planilha.

Para explanação de um dos objetivos, foi realizada uma consulta para analisar a distribuição de redes assistenciais de cardiologia pelo Brasil que atendem pelo sistema SUS.

Tabela 5 - Rede assistencial de cardiologia sistema SUS

ESTADO	QUANTIDADE
AC /	53
AL /	330
AM /	159
AP /	45
BA /	1360
CE /	419
DF /	304
ES /	423
GO /	780
MA /	230
MS /	291
PA /	320
PB /	349
PE /	1212
PI /	199
PR /	1401
RJ /	1953
RN /	338
RO /	116
RR /	43 **
RS /	1616
SC /	912
SE /	431
SP /	6151 *
TO /	118

Fonte: Elaborado pelo autor (2017)

É notório observar que existe uma desproporcionalidade no que se diz respeito a distribuição da rede assistencial médica no Brasil, o caso analisado em questão mostra que o estado de Roraima possui o menor quantitativo de disponibilidade de assistência cardiológica do Brasil, informação essa que fortalece a notícia divulgada segundo no site G1 - Roraima,

em que afirma que o estado possui 1,56 médicos para cada mil habitantes. Por outro lado, São Paulo lidera com o maior número de assistência cardiológica do país. Com essas informações em mãos pode-se explicar, por exemplo, a grande migração de pessoas para São Paulo em busca de soluções médicas e tentar explicar o que leva alguns estados indicados terem uma baixa oferta de especialidades médicas.

7. RESULTADOS

Para avaliação da usabilidade da solução foi utilizado *System Usability Scale* (SUS), uma das mais conhecidas e simples metodologia de averiguação do nível de usabilidade de um sistema.

Para Nielsen (2003), usabilidade é um atributo de qualidade que avalia a facilidade de uso de uma interface, sendo definida por cinco componentes:

1. Capacidade de aprendizagem: a facilidade de utilizar o sistema pela primeira vez;
2. Eficiência: rapidez para executar as tarefas;
3. Memorização: o processo de lembrar como utilizar o sistema, após um tempo sem utilizar;
4. Erros: ausência de erros apresentados pelo sistema;
5. Satisfação: design agradável

Segundo o *usability.gov* o SUS (*System Usability Scale*) consiste em um questionário de 10 itens com cinco opções de respostas para os entrevistados; originalmente criado por John Brooke em 1986. Ele permite que você avalie uma ampla variedade de produtos e serviços, incluindo *hardware*, *software*, dispositivos móveis, sites e aplicativos.

Para cada item do questionário o usuário pode responder em uma escala de 1 a 5, onde 1 significa Discordo Fortemente e 5 significa Concordo Fortemente.

Características que o SUS nos ajuda a avaliar:

- Efetividade (os usuários conseguem completar seus objetivos?)
- Eficiência (quanto esforço e recursos são necessários para isso?)
- Satisfação (a experiência foi satisfatória?)

7.1. Resultados do questionário

A ferramenta foi apresentada para professores, pesquisadores da área de computação e posteriormente foi aplicado o questionário SUS com os seguintes itens:

1. Gostaria de usar esta ferramenta com frequência.
2. A ferramenta é desnecessariamente complexa.
3. Achei a ferramenta fácil de usar.
4. Acho que precisaria do apoio de técnico para ser capaz de usar esta ferramenta.
5. Achei que as várias funções desta ferramenta foram bem integradas.

6. Achei que havia muita inconsistência nesta ferramenta.
7. Imagino que a maioria das pessoas aprenderá a usar este ferramenta muito rapidamente.
8. Achei a ferramenta muito complicado de usar.
9. Senti-me muito confiante usando a ferramenta.
10. Eu preciso aprender um monte de coisas antes de usar esta ferramenta

De acordo com Tenório et al. (2011) é notório o reconhecimento dos componentes de qualidade indicados nas questões do SUS:

- Facilidade de aprendizagem: 3, 4, 7 e 10;
- Eficiência: 5, 6 e 8;
- Facilidade de memorização: 2;
- Minimização dos erros: 6;
- Satisfação: 1, 4, 9.

O resultado da SUS é calculado somando a contribuição individual de cada item. Para os itens ímpares subtrai-se 1 à resposta do usuário, e para os itens pares o score é 5 menos a resposta do usuário. Após a obtenção do score de cada item, somam-se os scores e multiplica-se o resultado por 2,5 (BROOKE, 1986). Sendo assim, o score obtido será um índice de satisfação do utilizador (que varia de 0 a 100). De acordo com Sauro (2009), a pontuação SUS média dos 500 estudos que realizou foi de 68 pontos e Bangor et al. (2009) relatam que a média de 70 pontos tem se mantido em diferentes aplicações da SUS.

As figuras abaixo mostram a resposta média obtida (11 usuários no total) para cada pergunta dos questionários SUS sobre a usabilidade da ferramenta de pré-processamento e visualização de dados do DATASUS:

Figura 29 - Resposta média para pergunta 1



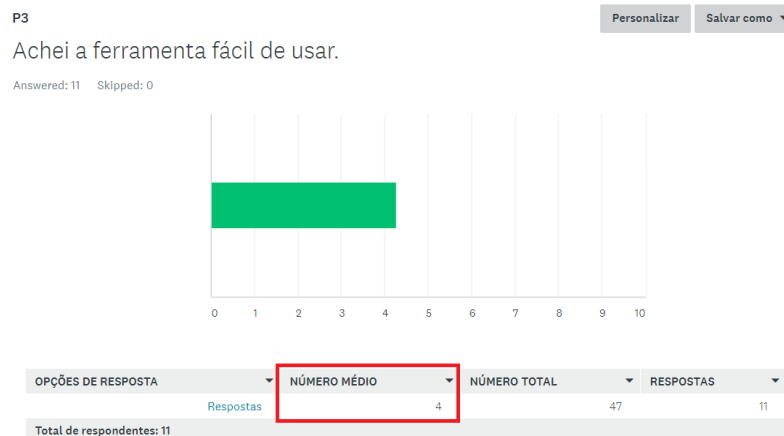
Fonte: < <https://pt.surveymonkey.com> >

Figura 30 - Resposta média para pergunta 2.



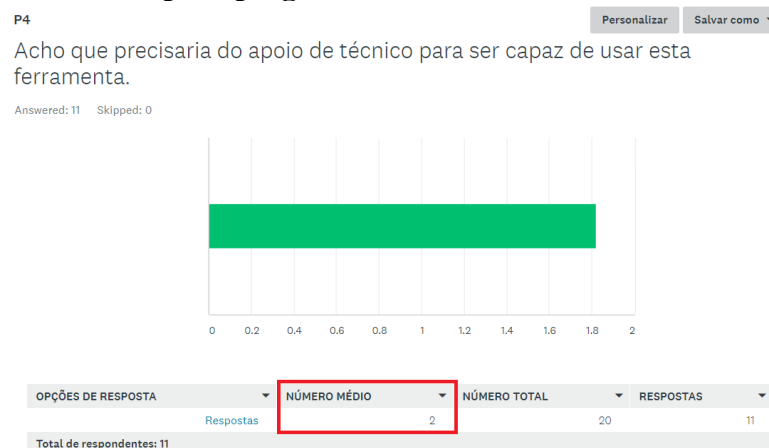
Fonte: <<https://pt.surveymonkey.com>>

Figura 31 - Resposta média para pergunta 3



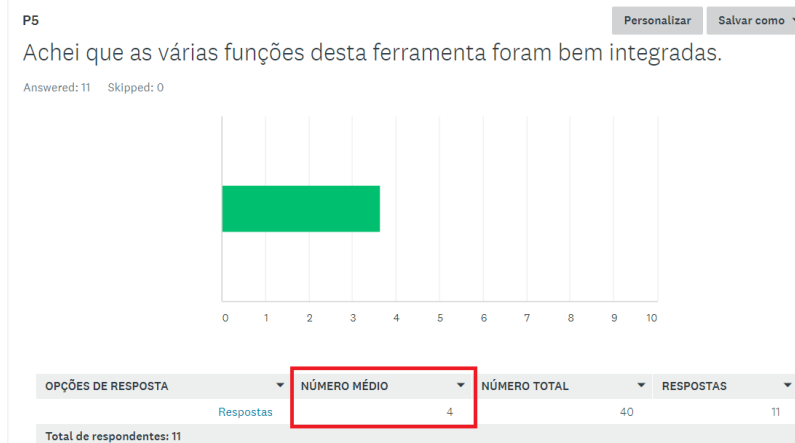
Fonte: <<https://pt.surveymonkey.com>>

Figura 32 - Resposta média para pergunta 4



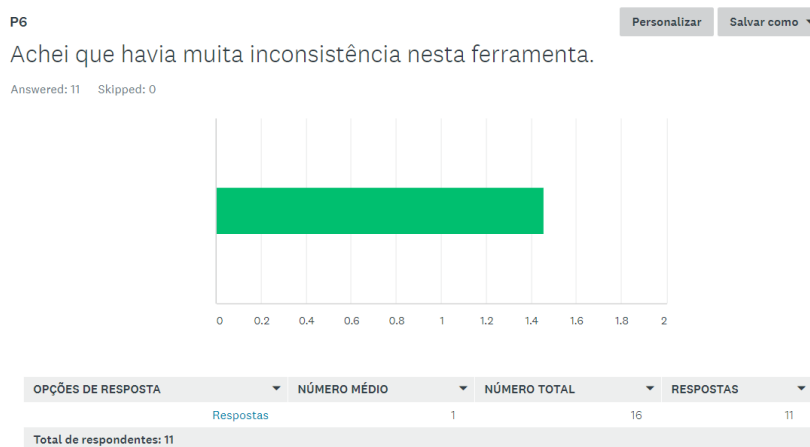
Fonte: <<https://pt.surveymonkey.com>>

Figura 33 - Resposta média para pergunta 5.



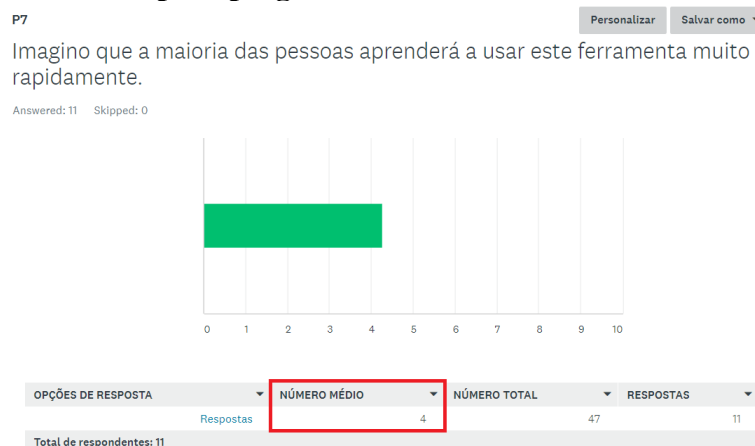
Fonte: <<https://pt.surveymonkey.com>>

Figura 34 - Resposta média para pergunta 6



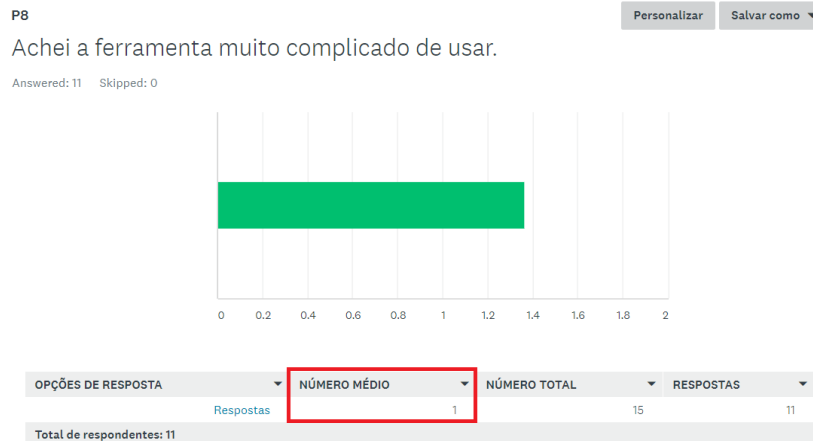
Fonte: <<https://pt.surveymonkey.com>>

Figura 35 - Resposta média para pergunta 7



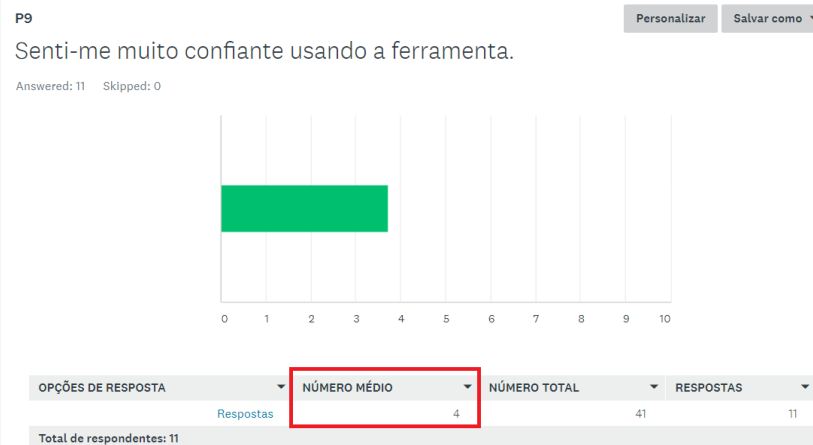
Fonte: <<https://pt.surveymonkey.com>>

Figura 36 - Resposta média para pergunta 8



Fonte: <<https://pt.surveymonkey.com>>

Figura 37 - Resposta média para pergunta 9



Fonte: <<https://pt.surveymonkey.com>>

Figura 38 - Resposta média para pergunta 10.



Fonte: <<https://pt.surveymonkey.com>>

Como próximo passo foi aplicado o cálculo de score subtraindo 1 da média de cada resposta ímpar e subtraindo de 5 da média de cada resposta par, obtendo-se a tabela de score abaixo:

Tabela 6 - Cálculo de score para perguntas ímpares

Média das respostas ímpares	Score SUS
P1 = 3	2
P3 = 4	3
P5 = 4	3
P7 = 4	3
P9 = 4	3
Total	14

Fonte: Elaborado pelo autor (2017)

Tabela 7 - Cálculo de score para perguntas pares

Média das respostas pares	Score SUS
P2 = 2	3
P4 = 2	3
P6 = 1	4
P8 = 1	4
P10 = 2	3
Total	17

Fonte: Elaborado pelo autor (2017)

Para obtenção da média final do score SUS foi feito a soma do scores das perguntas pares com os scores das perguntas ímpares ($14 + 17 = 31$) multiplicado por 2,5 obtendo assim o score final de 77,5.

A média do *System Usability Score* é 68 pontos, se o score obtido estiver abaixo dessa média, provavelmente o *software* está enfrentando problemas sérios de usabilidade, mas para a ferramenta desenvolvida neste trabalho o score resultante da aplicação do SUS ficou acima da média, alcançando o valor de 77,5 demonstrando que a ferramenta atende as exigências de usabilidade dos interessados em utilizá-la.

Embora a usabilidade tenha sido avaliada de forma positiva, este estudo ajudou ainda identificando oportunidades de melhoria do sistema, as quais contribuirão para uma maior usabilidade.

8. CONCLUSÃO

O DATASUS possui uma enorme base de dados com os mais variados tipos de informações, dentre elas: (indicadores de saúde, informações epidemiológicas e de morbidade, assistência à saúde, estatísticas vitais, informações sobre a rede de assistência à saúde, informações demográficas e socioeconômicas) além de dados financeiros (referentes aos recursos do Fundo Nacional de Saúde transmitidos aos municípios, aos prestadores de serviços de saúde, aos orçamentos públicos de saúde declarados pelos Estados, pelo Distrito Federal e pelos Municípios), mas essas informações são pouco exploradas.

Apesar de serem públicos, a captura, interpretação e exploração desses dados não é tão trivial quanto parece, necessitando de utilização de ferramentas disponibilizadas pelo DATASUS, as quais possuem vários obstáculos e limitações técnicas, por exemplo: só podem ser executadas em ambiente WINDOWS, não são *opensource*, portanto não podem ser adaptadas de acordo com cada necessidade de quem usa e consultas muito limitadas.

A ferramenta proposta neste trabalho visa diminuir esses obstáculos e limitações, oferecendo a democratização dessas informações de uma forma mais clara e simples. Por ser *opensource* se apresenta como uma ótima opção para ser aperfeiçoada e até adaptada para interesses semelhantes ao proposto neste trabalho, podendo ser executada em qualquer sistema operacional e com o desempenho ainda maior, como foi comparado no tópico 2.

Uma das suas maiores qualidades está na velocidade de captura,criptografia e armazenamento dessas informações em um banco de dados relacional ou planilhas, permitindo agora estudos na área da saúde que antes eram difíceis ou até impossíveis de serem realizados.

Mostrou-se ainda que a ferramenta alcançou um bom resultado na aplicação de um questionário usando a escala SUS (*System Usability Scale*) com a finalidade de avaliar a sua usabilidade, alcançando-se um score de 77,5 demonstrando que a ferramenta é efetiva, eficiente e satisfatória segundo a escala SUS.

REFERÊNCIAS

ADLER, M. **Decompressor for output of PKWare Data Compression Library (DCL)**, 10 jul. 2016. Disponível em: <<https://github.com/madler/zlib/tree/master/contrib/blast>>. Acesso em: 27 dez. 2017.

BARBOSA, H. P. **The importance of Digital TV for Countries in Development: a case study of Brazil.** iDTV. Tampere: [s.n.]. 2010.

BRASIL, M. D. S. D. **Informações de Saúde (TABNET).** DATASUS. Disponível em: <<http://www2.datasus.gov.br/DATASUS/index.php?area=02>>. Acesso em: 01 dez. 2017.

BROOKE, J. **SUS - A quick and dirty usability scale.** 1986. Disponível em <<https://hell.meiert.org/core/pdf/sus.pdf> >. Acesso em: 08 jan. 2019.

CNES. (s.d.). **CNES - Cadastro Nacional de Estabelecimentos de Saúde.** Disponível em: <<http://datasus.saude.gov.br/sistemas-e-aplicativos/cadastros-nacionais/cnes>>. Acesso em: 01 nov. 2018

CRIBARI-NETO, **Francisco. Algumas considerações sobre computação científica,** Julho 2013. Disponível em <http://conteudo.icmc.usp.br/CMS/Arquivos/arquivos_enviados/SECAO-POSGRAD_87_cribari-coputacao.pdf>. Acesso em: 15 dez. 2018

DEVMEDIA. **Acesso ao código nativo usando JNA (Java Native Access).** Disponível em: <<https://www.devmedia.com.br/acesso-ao-codigo-nativo-usando-jna-java-native-access/26535>>. Acesso em: 20 dez. 2017.

DOBORUVIKNE, D. (2017). **Java Native Access.** Disponível em: <<https://github.com/java-native-access/jna>>. Acesso em: 01 nov. 2018

DRISCOLL, M. **The 9 Best Languages For Crunching Data. FastCompany.** Disponível em: <<https://www.fastcompany.com/3030716/the-9-best-languages-for-crunching-data>>. Acesso em: 11 jan. 2018.

FERNÁNDEZ, A. **JavaDBF,** 15 jan. 2017. Disponível em: <<https://github.com/albfernandez/javadbfs>>. Acesso em: 28 dez. 2017.

G1-Roraima. (s.d.). **Número de médicos em Roraima é 29% menor que a média nacional.** Disponível em: <<https://g1.globo.com/rr/roraima/noticia/numero-de-medicos-em-roraima-e-29-menor-que-a-media-nacional-diz-crm-rr.ghtml>>. Acesso em: 15 nov. 2018

HORNIK, K. (s.d.). **What is R?** Disponível em: <<https://cran.r-project.org/doc/FAQ/R-FAQ.html>>. Acesso em: 20 nov. 2018.

JAVA.COM. **Obtenha Informações sobre a Tecnologia Java.** Disponível em: <https://www.java.com/pt_BR/about/>. Acesso em: 27 dez. 2017.

JONES, C. **Software Engineering Best Practices: lessons from successful projects in the top companies.** 2a.ed. [S.l.]: McGraw-Hill, 2009. 537–550p

ORACLE. **JAVA Native Interface Specification-Contents.** Disponível em: <<https://docs.oracle.com/javase/7/docs/technotes/guides/jni/spec/jniTOC.html>>. Acesso em: 15 dez. 2017.

_____. **JAVA Native Interface Specification-Contents.** Disponível em: <<https://docs.oracle.com/javase/7/docs/technotes/guides/jni/spec/jniTOC.html>>. Acesso em: 15 dez. 2018.

PETRUZALEK, D. **READ.DBC - Um Pacote para importação de dados do DATASUS na linguagem R,** Goiânia, 2016.

PKWARE. **PKWARE Data Compression Library.** Disponível em: <<https://support.pkware.com/display/PKZIP/DCL>>. Acesso em: 24 dez. 2017.

PM, F. **Code to convert from dbc to dbf [Internet].** Disponível em: <<https://github.com/eaglebh/blast-dbf>>. Acesso em: 26 dez. 2017.

Portal da Saúde. (s.d.). **Indicadores de Saúde e Pactuações.** Disponível em: <<http://www2.datasus.gov.br/DATASUS/index.php?area=0201>>. Acesso em: 03 nov. 2018.

ResearchGate. (s.d.). **Performance comparison between Java and JNI for optimal implementation of computational micro-kernels.** Disponível em: <https://www.researchgate.net/publication/269935434_Performance_comparison_between_Java_and_JNI_for_optimal_implementation_of_computational_micro-kernels>. Acesso em: 03 nov. 2018.

SAURO, Jeff. **Measuring Usability With The System Usability Scale (SUS).** 2009. Disponível em: <<https://www.userfocus.co.uk/articles/measuring-usability-with-the-SUS.html>>. Acesso em: 08 out. 2019.

Secretaria da Saúde. (s.d.). **Sanitária - Serviços - Estabelecimentos de Saúde.** Disponível em: <<http://www.saude.pr.gov.br/modules/conteudo/conteudo.php?conteudo=665>>. Acesso em: 01 nov. 2018.

TENÓRIO, Josceli Maria et al. **Desenvolvimento e Avaliação de um Protocolo Eletrônico para Atendimento e Monitoramento do Paciente com Doença Celíaca.** 2011. Disponível em: <https://seer.ufrgs.br/rita/article/view/rita_v17_n2_p210>. Acesso em: 7 jan. 2019.

USABILITY.GOV. **System Usability Scale (SUS)**. Disponível em: <<https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html>>. Acesso em: 5 jan. 2019

VANIUS ZAPALOWSKI, **Análise quantitativa e comparativa de linguagens de programação**, Porto Alegre, 2012.

VIANA, R. R. **Simplificando o acesso a código nativo com JNA**. MundoJ, n. 059, p. 30, 2016.

APÊNDICE

APÊNDICE A – TRECHO DE CÓDIGO EM JAVA COM A UTILIZAÇÃO DA BIBLIOTECA *DBCTODBF-1.1.JAR*

```

import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.URL;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.ArrayList;

import utils.ConexaoPostgresql;
import win.CBlastLibrary;

public class Main {
    private static String pathDBF = "";
    private static String pathDBC = "";

    private static String EXTENSAO_DBF = ".DBF";
    private static String EXTENSAO_DBC = ".DBC";

    public static void main(String[] args) {
        String dominio = "CNES";
        String tipoArquivo = "ST";
        String siglaEstado = "MA";
        String ano = "16";
        String mes = "01";
        String nomeArquivo = tipoArquivo + siglaEstado + ano + mes;
        String urlDownload = "ftp://ftp.datasus.gov.br/dissemin/publicos/CNES/200508_/Dados/" +
tipoArquivo + "/"
                                + nomeArquivo + EXTENSAO_DBC;
        String tempDir = System.getProperty("java.io.tmpdir");
        System.out.println("OS current temporary directory is " + tempDir);
        double cont = 0;
        File fileDBC = new File("DADOS" + File.separatorChar + dominio + File.separatorChar +
tipoArquivo
                                + File.separatorChar + nomeArquivo + EXTENSAO_DBC);
        File fileDBF = null;
        ConexaoPostgresql banco = new ConexaoPostgresql();
        banco.conectar();
        DatasusDBFReader dbf = null;

        try {
            System.out.println("##### REALIZANDO DOWNLOAD: "
+ urlDownload);
            downloadUsingStream(urlDownload, fileDBC);

            pathDBF = tempDir + nomeArquivo + EXTENSAO_DBF;
            fileDBF = new File(pathDBF);
            // System.out.println("PATH DBF: " + tempDir);

            // System.out.println("##### CRIANDO ARQUIVO .DBF
EM PASTA
            // TEMPORoRIA: "+pathDBF);
            CBlastLibrary cLib = new CBlastLibrary();

```

```

long tempoInicial = System.currentTimeMillis();
System.out.println(fileDBC.getAbsolutePath());

cLib.dbc2dbf(fileDBC.getAbsolutePath(), pathDBF);
cont = (System.currentTimeMillis() - tempoInicial) / 1000.00;
// System.out.println("o metodo executou em " + cont);

// System.out.println("##### LENDO DADOS DO
ARQUIVO:
// "+pathDBF);
dbf = new DatasusDBFReader(pathDBF);
cont = (System.currentTimeMillis() - tempoInicial) / 1000.00;
System.out.println("o metodo executou em " + cont);

Connection con = banco.getCon();

System.out.println("##### INSERINDO DADOS LIDOS
DO .DBF NO BANCO DE DADOS...");
dbWrite(dominio, tipoArquivo, dbf, banco);
banco.getCon().commit();
banco.desconectar();
cont = (System.currentTimeMillis() - tempoInicial) / 1000.00;
// System.out.println("o metodo executou em " + cont);

} catch (IOException e) {
System.err.println("ERRO AO CRIAR/BAIXAR ARQUIVO .DBC: " +
fileDBC.getAbsolutePath()
+ " , VERIFIQUE SE O CAMINHO EXISTE.");
fileDBC.delete();
dbf.close();
fileDBF.delete();
e.printStackTrace();
} catch (SQLException e) {
try {
banco.getCon().rollback();
fileDBC.delete();
dbf.close();
fileDBF.delete();
e.printStackTrace();
} catch (SQLException e1) {
System.err.println("ERRO AO DAR COMMIT/ROLLBACK NA
TRANSAÇÃO.");
fileDBC.delete();
dbf.close();
fileDBF.delete();
e1.printStackTrace();
}
} catch (Exception e) {
// TODO: handle exception
}

System.out.println("##### FINALIZADO #####");
}

public static void dbWrite(String dominio, String tipoArquivo, DatasusDBFReader dbf,
ConexaoPostgresql banco)
throws SQLException {

if (dbf == null) {

```

```

        return;
    }

    String cabecalhos = dbf.cabecalhos();
    ArrayList<String> linhas = dbf.linhas();

    String comandoCreate = "CREATE TABLE IF NOT EXISTS " + dominio + "_" + tipoArquivo
+ " (" + cabecalhos + ")";

    // System.out.println(comandoCreate);

    banco.getStmt().execute(comandoCreate.toLowerCase());

    System.out.println("Total de linhas:" + linhas.size());

    for (String linha : linhas) {
        // System.out.println(linha);
        String comandoInsert = "INSERT INTO " + dominio + "_" + tipoArquivo + "
VALUES (" + linha + ")";
        banco.getStmt().execute(comandoInsert);
    }
}

private static void downloadUsingStream(String urlStr, File fileDBC) throws IOException {
    String caminhoArquivoDBC = "";

    fileDBC.delete();
    fileDBC.createNewFile();
    caminhoArquivoDBC = fileDBC.getAbsolutePath();
    System.out.println("Caminho do JDC baixado: ----- " + caminhoArquivoDBC);
    URL url = new URL(urlStr);

    InputStream stream = url.openStream();
    BufferedInputStream bis = new BufferedInputStream(stream);
    FileOutputStream fis = new FileOutputStream(fileDBC);
    byte[] buffer = new byte[1024];
    int count = 0;
    System.out.println("CRIANDO ARQUIVO: " + caminhoArquivoDBC);

    while ((count = bis.read(buffer, 0, 1024)) != -1) {
        fis.write(buffer, 0, count);
    }
    fis.close();
    bis.close();
}
}

```

ANEXOS

ANEXO I - BLAST.C

```

/* blast.c
 * Copyright (C) 2003, 2012 Mark Adler
 * For conditions of distribution and use, see copyright notice in blast.h
 * version 1.2, 24 Oct 2012
#include <setjmp.h>      /* for setjmp(), longjmp(), and jmp_buf */
#include "blast.h"      /* prototype for blast() */

#define MAXBITS 13      /* maximum code length */
#define MAXWIN 4096     /* maximum window size */

/* input and output state */
struct state {
    /* input state */
    blast_in infun;      /* input function provided by user */
    void *inhow;        /* opaque information passed to infun() */
    unsigned char *in;  /* next input location */
    unsigned left;      /* available input at in */
    int bitbuf;         /* bit buffer */
    int bitcnt;         /* number of bits in bit buffer */

    /* input limit error return state for bits() and decode() */
    jmp_buf env;

    /* output state */
    blast_out outfun;    /* output function provided by user */
    void *outhow;       /* opaque information passed to outfun() */
    unsigned next;      /* index of next write location in out[] */
    int first;          /* true to check distances (for first 4K) */
    unsigned char out[MAXWIN]; /* output buffer and sliding window */
};

static int bits(struct state *s, int need)
{
    int val;            /* bit accumulator */

    /* load at least need bits into val */
    val = s->bitbuf;
    while (s->bitcnt < need) {
        if (s->left == 0) {
            s->left = s->infun(s->inhow, &(s->in));
            if (s->left == 0) longjmp(s->env, 1); /* out of input */
        }
        val |= (int)(*(s->in)++) << s->bitcnt; /* load eight bits */
        s->left--;
        s->bitcnt += 8;
    }

    /* drop need bits and update buffer, always zero to seven bits left */
    s->bitbuf = val >> need;
    s->bitcnt -= need;

    /* return need bits, zeroing the bits above that */
    return val & ((1 << need) - 1);
}

```

```

struct huffman {
    short _count;    /* number of symbols of each length */
    short _symbol; /* canonically ordered symbols */
};

static int decode(struct state _s, struct huffman *h)
{
    int len;          /* current number of bits in code */
    int code;        /* len bits being decoded */
    int first;       /* first code of length len */
    int count;       /* number of codes of length len */
    int index;       /* index of first code of length len in symbol table */
    int bitbuf;      /* bits from stream */
    int left;        /* bits left in next or left to process */
    short _next;    /* next number of codes */

    bitbuf = s->bitbuf;
    left = s->bitcnt;
    code = first = index = 0;
    len = 1;
    next = h->count + 1;
    while (1) {
        while (left--) {
            code |= (bitbuf & 1) ^ 1; /* invert code */
            bitbuf >>= 1;
            count = _next++;
            if (code < first + count) { /* if length len, return symbol */
                s->bitbuf = bitbuf;
                s->bitcnt = (s->bitcnt - len) & 7;
                return h->symbol[index + (code - first)];
            }
            index += count; /* else update for next length */
            first += count;
            first <<= 1;
            code <<= 1;
            len++;
        }
        left = (MAXBITS+1) - len;
        if (left == 0) break;
        if (s->left == 0) {
            s->left = s->infun(s->inhow, &(s->in));
            if (s->left == 0) longjmp(s->env, 1); /* out of input */
        }
        bitbuf = (s->in)++;
        s->left--;
        if (left > 8) left = 8;
    }
    return -9; /* ran out of codes */
}

static int construct(struct huffman *h, const unsigned char *rep, int n)
{
    int symbol; /* current symbol when stepping through length[] */
    int len;    /* current length when stepping through h->count[] */
    int left;   /* number of possible codes left of current length */
    short offs[MAXBITS+1]; /* offsets in symbol table for each length */
    short length[256]; /* code lengths */

    /* convert compact repeat counts into symbol bit length list */
    symbol = 0;

```



```

do {
    len = *rep++;
    left = (len >> 4) + 1;
    len &= 15;
    do {
        length[symbol++] = len;
    } while (--left);
} while (--n);
n = symbol;

/* count number of codes of each length */
for (len = 0; len <= MAXBITS; len++)
    h->count[len] = 0;
for (symbol = 0; symbol < n; symbol++)
    (h->count[length[symbol]])++; /* assumes lengths are within bounds */
if (h->count[0] == n) /* no codes! */
    return 0; /* complete, but decode() will fail */

/* check for an over-subscribed or incomplete set of lengths */
left = 1; /* one possible code of zero length */
for (len = 1; len <= MAXBITS; len++) {
    left <<= 1; /* one more bit, double codes left */
    left -= h->count[len]; /* deduct count from possible codes */
    if (left < 0) return left; /* over-subscribed--return negative */
} /* left > 0 means incomplete */

/* generate offsets into symbol table for each length for sorting */
offs[1] = 0;
for (len = 1; len < MAXBITS; len++)
    offs[len + 1] = offs[len] + h->count[len];

/*
 * put symbols in table sorted by length, by symbol order within each
 * length
 */
for (symbol = 0; symbol < n; symbol++)
    if (length[symbol] != 0)
        h->symbol[offs[length[symbol]]++] = symbol;

/* return zero for complete set, positive for incomplete set */
return left;
}

static int decomp(struct state *s)
{
    int lit; /* true if literals are coded */
    int dict; /* log2(dictionary size) - 6 */
    int symbol; /* decoded symbol, extra bits for distance */
    int len; /* length for copy */
    unsigned dist; /* distance for copy */
    int copy; /* copy counter */
    unsigned char *from, *to; /* copy pointers */
    static int virgin = 1; /* build tables once */
    static short litcnt[MAXBITS+1], litsym[256]; /* litcode memory */
    static short lencnt[MAXBITS+1], lensym[16]; /* lencode memory */
    static short distcnt[MAXBITS+1], distsym[64]; /* distcode memory */
    static struct huffman litcode = {litcnt, litsym}; /* length code */
    static struct huffman lencode = {lencnt, lensym}; /* length code */
    static struct huffman distcode = {distcnt, distsym}; /* distance code */
    /* bit lengths of literal codes */

```

```

static const unsigned char litlen[] = {
    11, 124, 8, 7, 28, 7, 188, 13, 76, 4, 10, 8, 12, 10, 12, 10, 8, 23, 8,
    9, 7, 6, 7, 8, 7, 6, 55, 8, 23, 24, 12, 11, 7, 9, 11, 12, 6, 7, 22, 5,
    7, 24, 6, 11, 9, 6, 7, 22, 7, 11, 38, 7, 9, 8, 25, 11, 8, 11, 9, 12,
    8, 12, 5, 38, 5, 38, 5, 11, 7, 5, 6, 21, 6, 10, 53, 8, 7, 24, 10, 27,
    44, 253, 253, 253, 252, 252, 252, 13, 12, 45, 12, 45, 12, 61, 12, 45,
    44, 173};
    /* bit lengths of length codes 0..15 */
static const unsigned char lenlen[] = {2, 35, 36, 53, 38, 23};
    /* bit lengths of distance codes 0..63 */
static const unsigned char distlen[] = {2, 20, 53, 230, 247, 151, 248};
static const short base[16] = { /* base for length codes */
    3, 2, 4, 5, 6, 7, 8, 9, 10, 12, 16, 24, 40, 72, 136, 264};
static const char extra[16] = { /* extra bits for length codes */
    0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 5, 6, 7, 8};

    /* set up decoding tables (once--might not be thread-safe) */
if (virgin) {
    construct(&litcode, litlen, sizeof(litlen));
    construct(&lencode, lenlen, sizeof(lenlen));
    construct(&distcode, distlen, sizeof(distlen));
    virgin = 0;
}

    /* read header */
    lit = bits(s, 8);
if (lit > 1) return -1;
    dict = bits(s, 8);
if (dict < 4 || dict > 6) return -2;

    /* decode literals and length/distance pairs */
do {
    if (bits(s, 1)) {
        /* get length */
        symbol = decode(s, &lencode);
        len = base[symbol] + bits(s, extra[symbol]);
        if (len == 519) break; /* end code */

        /* get distance */
        symbol = len == 2 ? 2 : dict;
        dist = decode(s, &distcode) << symbol;
        dist += bits(s, symbol);
        dist++;
        if (s->first && dist > s->next)
            return -3; /* distance too far back */

        /* copy length bytes from distance bytes back */
        do {
            to = s->out + s->next;
            from = to - dist;
            copy = MAXWIN;
            if (s->next < dist) {
                from += copy;
                copy = dist;
            }
            copy -= s->next;
            if (copy > len) copy = len;
            len -= copy;
            s->next += copy;
        } do {

```

```

        *to++ = *from++;
    } while (--copy);
    if (s->next == MAXWIN) {
        if (s->outfun(s->outhow, s->out, s->next)) return 1;
        s->next = 0;
        s->first = 0;
    }
    } while (len != 0);
}
else {
    /* get literal and write it */
    symbol = lit ? decode(s, &litcode) : bits(s, 8);
    s->out[s->next++] = symbol;
    if (s->next == MAXWIN) {
        if (s->outfun(s->outhow, s->out, s->next)) return 1;
        s->next = 0;
        s->first = 0;
    }
}
} while (1);
return 0;
}

/* See comments in blast.h */
int blast(blast_in infun, void *inhow, blast_out outfun, void *outhow)
{
    struct state s;          /* input/output state */
    int err;                /* return value */

    /* initialize input state */
    s.infun = infun;
    s.inhow = inhow;
    s.left = 0;
    s.bitbuf = 0;
    s.bitcnt = 0;

    /* initialize output state */
    s.outfun = outfun;
    s.outhow = outhow;
    s.next = 0;
    s.first = 1;

    /* return if bits() or decode() tries to read past available input */
    if (setjmp(s.env) != 0) /* if came back here via longjmp(), */
        err = 2;          /* then skip decomp(), return error */
    else
        err = decomp(&s); /* decompress */

    /* write any leftover output and update the error code if needed */
    if (err != 1 && s.next && s.outfun(s.outhow, s.out, s.next) && err == 0)
        err = 1;
    return err;
}

```

ANEXO II – BLAST.H

```
typedef unsigned (*blast_in)(void *how, unsigned char **buf);  
typedef int (*blast_out)(void *how, unsigned char *buf, unsigned len);  
int blast(blast_in infun, void *inhow, blast_out outfun, void *outhow);
```

ANEXO III – CERTIFICADO DE REGISTRO DE PROGRAMA DATASUSVIEWER



REPÚBLICA FEDERATIVA DO BRASIL
MINISTÉRIO DA ECONOMIA
INSTITUTO NACIONAL DA PROPRIEDADE INDUSTRIAL
DIRETORIA DE PATENTES, PROGRAMAS DE COMPUTADOR E TOPOGRAFIAS DE CIRCUITOS INTEGRADOS

Certificado de Registro de Programa de Computador

Processo Nº: **BR512019000042-0**

O Instituto Nacional da Propriedade Industrial expede o presente certificado de registro de programa de computador, válido por 50 anos a partir de 1º de janeiro subsequente à data de 10/12/2017, em conformidade com o §2º, art. 2º da Lei 9.609, de 19 de Fevereiro de 1998.

Título: datasusViewer - Sistema de Visualização de dados do DATASUS advindos de um banco de dados relacional

Data de criação: 10/12/2017

Titular(es): UNIVERSIDADE ESTADUAL DO MARANHÃO - UEMA

Autor(es): ANTONIO FERNANDO LAVAREDA JACOB JUNIOR; DIEGO PEREIRA MENDES

Linguagem: JAVA; SQL

Campo de aplicação: IF-01; SD-01

Tipo de programa: FA-04; GI-07; GI-08

Algoritmo hash: SHA-512

Resumo digital hash:

857c60e00d44ce0d8db5cf9de681bbf4501f216c4a45cd14c770b90c44100ea56eb400282988cfd77d141f51038ac374318
8f8ac421cc78fe4e059eb63d210e9

Expedido em: 15/01/2019

Aprovado por:

Alexandre Gomes Ciancio

Diretor Substituto de Patentes, Programas de Computador e Topografias de Circuitos Integrados

ANEXO IV – CERTIFICADO DE REGISTRO DE PROGRAMA DATASUSIMPORT

	 
	REPÚBLICA FEDERATIVA DO BRASIL MINISTÉRIO DA ECONOMIA INSTITUTO NACIONAL DA PROPRIEDADE INDUSTRIAL DIRETORIA DE PATENTES, PROGRAMAS DE COMPUTADOR E TOPOGRAFIAS DE CIRCUITOS INTEGRADOS
	Certificado de Registro de Programa de Computador
	Processo Nº: BR512019000041-2
	O Instituto Nacional da Propriedade Industrial expede o presente certificado de registro de programa de computador, válido por 50 anos a partir de 1º de janeiro subsequente à data de 10/12/2017, em conformidade com o §2º, art. 2º da Lei 9.609, de 19 de Fevereiro de 1998.
	Título: datasusImport - Sistema de Importação de dados do DATASUS para um banco de dados relacional
	Data de criação: 10/12/2017
	Titular(es): UNIVERSIDADE ESTADUAL DO MARANHÃO - UEMA
	Autor(es): ANTONIO FERNANDO LAVAREDA JACOB JUNIOR; DIEGO PEREIRA MENDES
	Linguagem: JAVA; SQL
Campo de aplicação: IF-01; SD-01	
Tipo de programa: FA-04; GI-07; GI-08	
Algoritmo hash: SHA-512	
Resumo digital hash: 7d24a484c1345a758133c1d69d16e95d5bce29aac310945772dad20b31286778a0c5c68e2447679dae203fd3d906140d6134191bdb8eec41450030bc97e3f095	
Expedido em: 15/01/2019	
	
Aprovado por: Alexandre Gomes Ciancio Diretor Substituto de Patentes, Programas de Computador e Topografias de Circuitos Integrados	