

Detecção de websites phishing com RDF e SPARQL

Mateus Santos
Universidade Federal do Rio de Janeiro

August 8, 2022

Abstract

Baseado no artigo de Muppavarapu, a ideia deste trabalho é construir uma aplicação capaz de identificar se um dado website pode ser classificado como Phishing. Para a execução desta tarefa é utilizado conceitos de desenvolvimento web e web semântica (RDF e SPARQL).

1 Introdução

Este projeto tem como propósito se inspirar no artigo de Muppavarapu para classificar websites Phishing, algo que vem se tornando cada vez mais presente no dia a dia.

Phishing é uma prática bem conhecida em que os invasores roubam informações confidenciais dos usuários (e às vezes dinheiro) falsificando sites ou induzindo os usuários a visitar algum site falso onde divulgam suas informações pessoais aos invasores, apesar de essas ações não serem intencionais e inocentes. Embora esses ataques não sejam novos na era da Internet, o phishing atraiu muita atenção nos últimos anos, interrompendo setores importantes como finanças, redes sociais, comércio eletrônico e causando enormes perdas econômicas, e os motivos são evidentes.

Quanto mais negócios acontecem na internet, mais dinheiro gira e mais oportunidades de ataque aparecem. As vítimas de ataques de phishing geralmente descobrem que suas informações pessoais ou financeiras foram roubadas sem o conhecimento do usuário. Detalhes de cartão de crédito, dados bancários pessoais, informações pessoais, perguntas de segurança, detalhes de login, etc. são os alvos desses ataques de phishing. Spear phishing é outro tipo de ataque de phishing no qual os invasores visam funcionários, como chefes militares e executivos de negócios, na tentativa de obter suas credenciais confidenciais.

De acordo com um relatório da empresa de segurança RivestShamir-Adleman (RSA), houve aproximadamente 61.278 ataques em novembro de 2014 [17], um aumento de 76% em relação a outubro de 2014. Isso se deve principalmente ao alto volume de compras online, especialmente durante o Natal período.

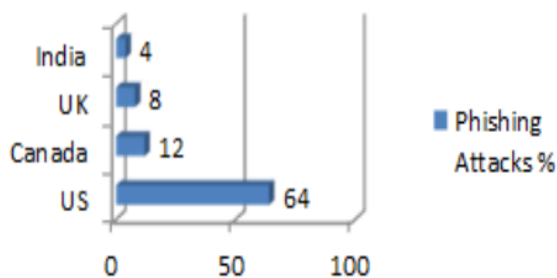


Figure 1: Estatísticas mostrando o volume de ataques em janeiro de 2015. Fonte: Artigo de Muppavarapu.

O artigo de Muppavarapu desenvolveu a seguinte metodologia de classificação de um website.

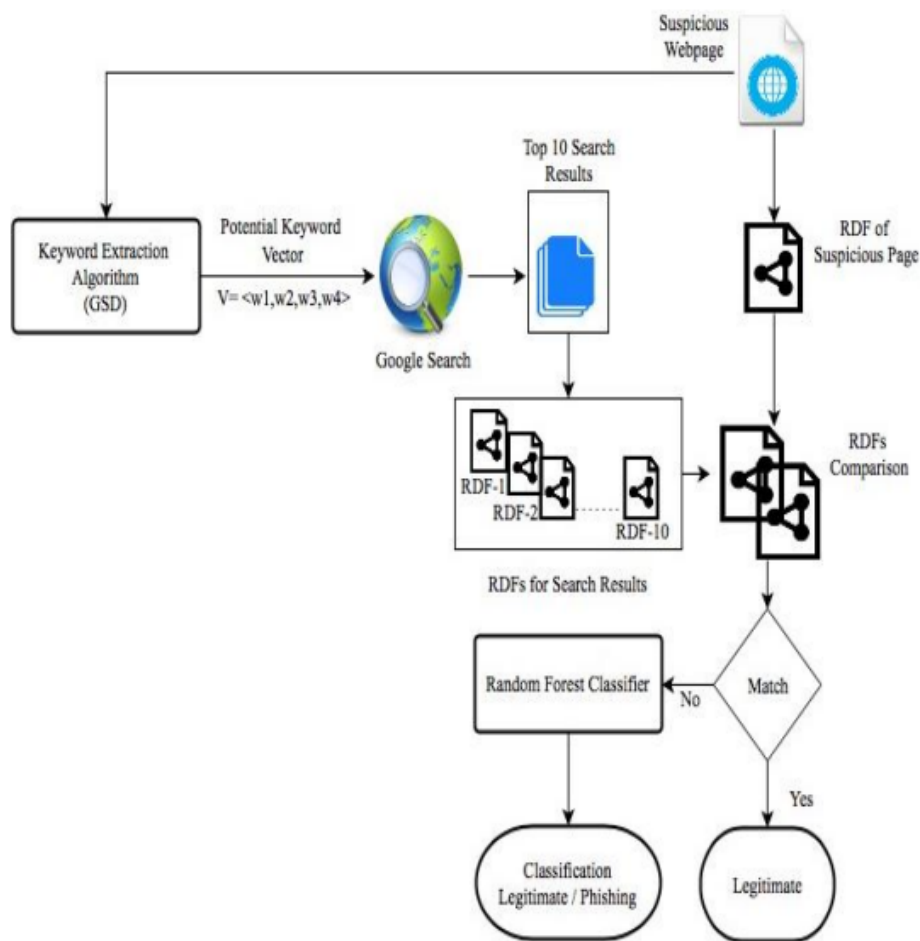


Figure 2: Diagrama do fluxo de classificação. Fonte: Artigo de Muppavarapu.

2 Objetivo

O intuito deste trabalho é construir uma aplicação web capaz de classificar se um dado website pode ser considerado como Phishing. A inspiração deste trabalho é o artigo tratado na subseção anterior, isto é, para a classificação do website será utilizada uma metodologia parecida com a do artigo. Dado um website o objetivo é extrair informações (metadados) do website e salvar no banco de dados GraphDB em forma de triplas RDF.

3 Metodologia

A ideia do projeto consiste dado um website, coletar informações do mesmo, isto é, extrair metadados e keywords do mesmo. Uma vez isto feito é pesquisado no DuckDuckGo estes termos para se obter os top resultados. Para cada um destes resultados é extraído os metadados e keywords da mesma forma que é feito para o website alvo. Uma vez isto feito, o modelo RDF de cada um dos websites é gerado usando informações de metadados dos mesmos, cada RDF de um website possui um id próprio que é a sua URL, após isso a informação é salva no GraphDB.

Uma vez tudo estando no GraphDB é feito uma query SPARQL para se buscar as informações do website alvo, com estas informações é feito outra query SPARQL para procurar se há algum outro website com estas informações, se sim, significa que o site apareceu no top de resultados do DuckDuckGo, e portanto o site é classificado como autêntico, caso contrário, é considerado como Phishing.

```
Querying GraphDB:
PREFIX ex:<http://example.org/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT * WHERE {
  ?s a foaf:Document .
  ?s ?p ?o .
  FILTER (?s = ex:target_www.saraiva.com.br)
}
LIMIT 100
```

Figure 3: Query SPARQL para buscar informações do site alvo (variáveis já substituídas).

```
Querying GraphDB:
PREFIX ex:<http://example.org/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT * WHERE {
  ?s rdf:type foaf:Document .
  ?s foaf:title "Livros, Games e mais | Saraiva" .
  ?s foaf:page "https://www.saraiva.com.br" .
  ?s ?p ?o .
  FILTER (?s != ex:target_www.saraiva.com.br)
}
LIMIT 100
```

Figure 4: Query SPARQL para buscar por grafos isomórficos (variáveis já substituídas).

Uma observação importante é que, por questões de complexidade, não foram utilizados todos os atributos que o artigo utilizou para identificar um website. A imagem a seguir mostra todos os atributos utilizados pelo artigo enquanto que as marcações em vermelho mostram os atributos utilizados neste projeto.

S.No	Properties	Description	Source
1	Title	Name or Title of the given web page.	Dublin Core
2	Parent Domain Name	Give the character set format of the web page.	Novel
3	Creator	An entity primarily responsible in creation of the web page. Examples of a Creator include a person or an organization.	Dublin Core
4	Subject	Typically, the subject will be represented using keywords, key phrases of the web page.	Dublin Core
5	Description	A brief description of the web page.	Dublin Core
6	Creation Date	Gives the creation date as well as the last modified date of the web page. Can be obtained from WhoIs lookup	Dublin Core
7	Identifier	URI of the web page.	Dublin Core
8	Status code	Gives the status code number that is returned from HTTP response.	HTTP
9	Form Action	Action tag of the form, checking this avoids XSS attacks	Novel
10	Form name	Name given to the form element	XHTML
11	Form method	GET or POST method of the form	Novel

12	Imgsrc	Gets the Source of the images	XHTML
13	Age of Domain	Gets the age of the domain by doing WhoIs lookup	Novel
14	Copyright	Gives the copyright information of the web page.	Novel
15	Captcha	Check for captcha in the web page	Novel
16	Frame src	Gets the source of the frames in web page	Novel
17	IP address	Gets the domains IP addresses of the web page	Novel
18	Updated date	Last Updated date of the Domain	Novel
19	Expiration Date	Expiration date of the Domain	Novel
20	Registrar	Registrar under which the Domain is registered	Novel

All the above mentioned features are represented as RDF properties each represented using a triplet (subject, predicate and object). These features extracted from the web page are represented as RDF statements forming RDF model for the page. Sometimes

Figure 5: Atributos para classificar um website, comparação artigo vs projeto (em vermelho).

4 Implementação

O sistema desenvolvido é uma aplicação web contendo um backend e frontend, a lógica de processamento responsável pela classificação se encontra no backend enquanto que o frontend foca nas questões de interface e interação com usuário. O backend foi implementado em Python e oferece uma API REST via biblioteca Flask, esta API é consultada pelo frontend escrito em Vue.js. O código fonte ?? se encontra hospedado no Github.

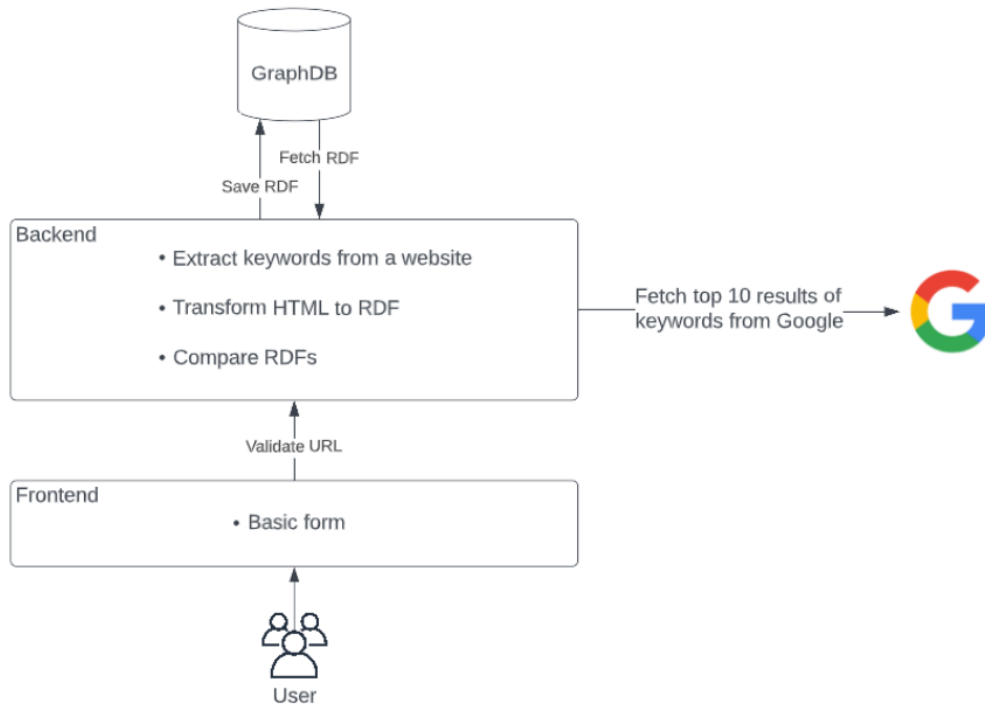


Figure 6: Arquitetura do sistema desenvolvido

O código principal do backend é dividido nas seguintes funções:

- **validate**: Código principal que chama as outras funções a fim de realizar a classificação
- **extract_metadata**: Extrai metadados e keywords de um website.
- **create_rdf**: Gera o modelo RDF Turtle.
- **save_rdf**: Salva o modelo RDF Turtle
- **query_sparql**: Realiza uma Query SPARQL no GraphDB.
- **check_for_isomorphic_graphs**: Verifica se dado um id, existe outro website com as mesmas informações.
- **search_duckduckgo**: Faz uma busca de termos no DuckDuckGo.

A imagem a seguir mostra o código-fonte da função principal do backend, a validate, para realizar a classificação do website alvo.

```
36 You, 56 minutes ago • add files
37 @blueprint.route('/validate', methods=['POST'])
38 def validate():
39     params = req.get_json()
40     target_url = params.get('url')
41     target_url = urlparse(target_url)
42
43     target_metadata = extract_metadata(f'{target_url.scheme}://{target_url.netloc}')
44
45     target_id = f'target_{target_url.netloc}'
46     rdf = create_rdf(target_id, target_metadata)
47
48 > graphdb_stmt = f'''...
52
53 # Prepare search
54 terms = []
55 > if('title' in target_metadata):...
57 > if('keywords' in target_metadata):...
59
60 search_result = search_duckduckgo(terms, max_results=3)
61
62 searched_domains = []
63 > for url in search_result:...
73
74 save_rdf(graphdb_stmt)
75
76 result = check_for_isomorphic_graphs(f'ex:{target_id}')
77
78 return jsonify({'url': target_url.netloc, 'rdf': graphdb_stmt, 'metadata': target_metadata, 'search_result': search_result, 'result': result})
```

Figure 7: Código-fonte da função validate.

O frontend do projeto consiste em apenas um campo para o usuário inputar a URL do website alvo e um formulário de resultados. A imagem a seguir ilustra a interface do frontend.

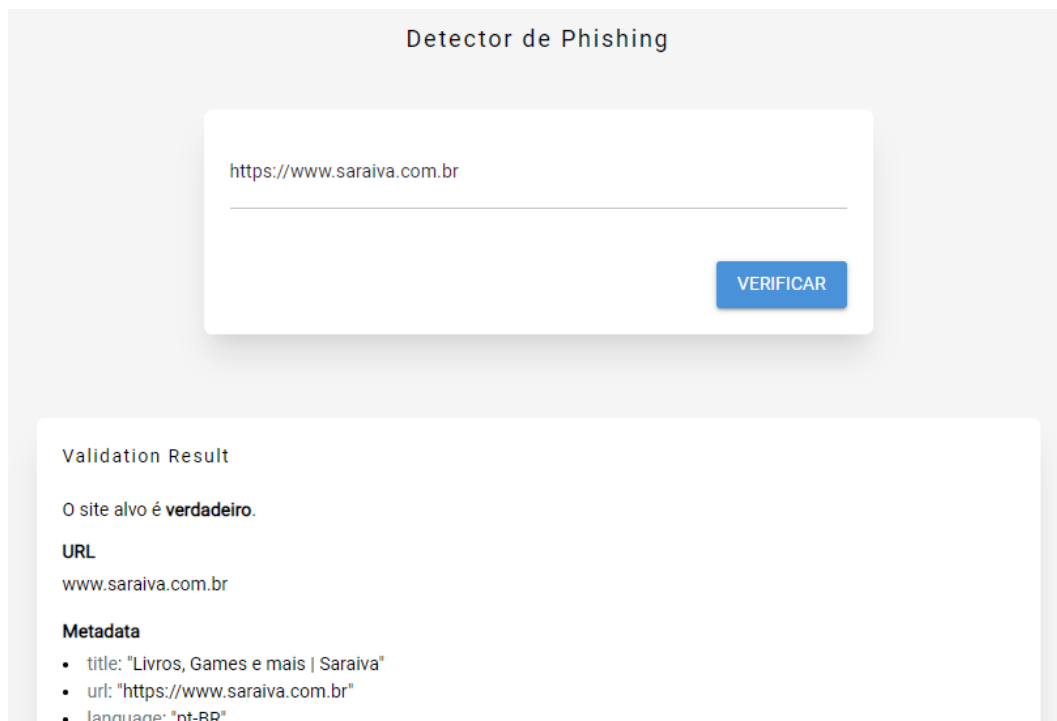


Figure 8: Imagem do frontend desenvolvido.

5 Obstáculos

Houveram muitos problemas no decorrer do desenvolvimento. Um problema inicial foi a instalação das dependências em Python, pois houve conflitos entre as bibliotecas, após um certo tempo o problema foi resolvido e as versões compatíveis foram registradas no arquivo backend/requirements.txt.

Outra problemática se deu na parte de fazer pesquisas no Google, pois o Google limita a quantidade de buscas que podem ser feitas via programação, após o limite é necessária pagar uma taxa que para o escopo deste projeto era inviável. A solução para isto foi usar o DuckDuckGo no lugar do Google.

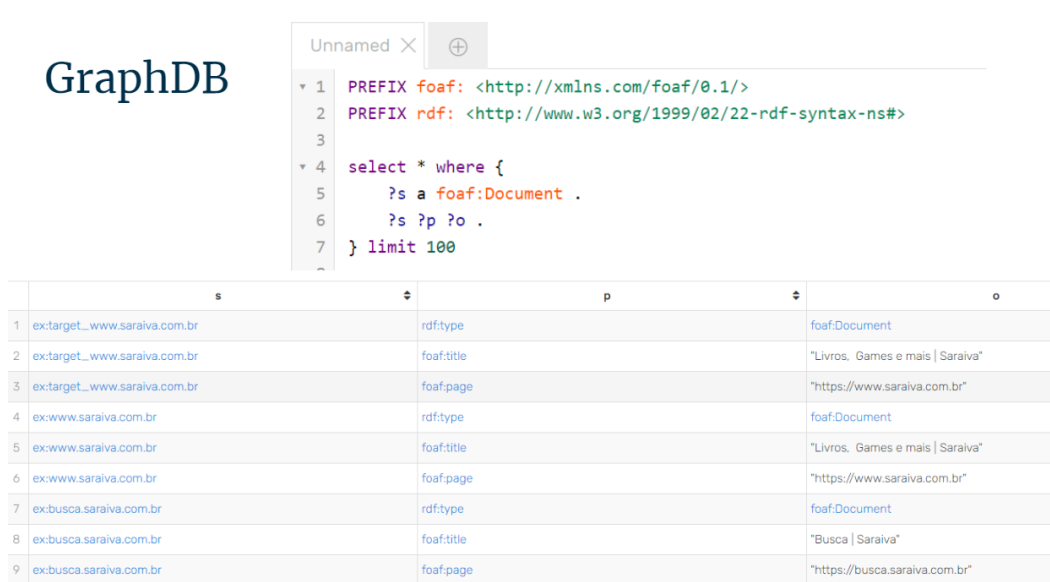
Houveram inúmeros outros obstáculos, porém outro marcante foi a obsolência da biblioteca python-graphdb para comunicação com o banco de dados. A documentação é antiga e faz referência para links que não existem mais. A documentação da API do GraphDB é confusa e difícil de fazer buscas. No final a biblioteca foi ignorada e as requests foram feitas diretamente para a API do GraphDB local.

```
File "C:\Users\mvsantos\AppData\Roaming\Python\Python38\site-packages\graphdb\rd
    raise ApiException(status=0, reason=msg)
graphdb.rdf4j.rest.ApiException: (0)
Reason: Cannot prepare a request message for provided
        arguments. Please check that your arguments match
        declared content type.
```

Figure 9: Um dos obstáculos, escrita no GraphDB. Apesar de seguir as instruções da API do GraphDB a mesma ainda retornava erros.

6 Resultados

Um fluxo completo, porém simples foi estabelecido com sucesso, a implementação ficou funcionável e é capaz de classificar, de uma forma mínima, um website.



The screenshot shows the GraphDB web interface. On the left, the 'GraphDB' logo is visible. The main area displays a query editor with a query named 'Unnamed'. The query is as follows:

```
1 PREFIX foaf: <http://xmlns.com/foaf/0.1/>
2 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
3
4 select * where {
5   ?s a foaf:Document .
6   ?s ?p ?o .
7 } limit 100
```

Below the query editor, a table of results is displayed. The table has three columns: 's' (subject), 'p' (predicate), and 'o' (object). The results are as follows:

s	p	o
ex:target_www.saraiva.com.br	rdf:type	foaf:Document
ex:target_www.saraiva.com.br	foaf:title	"Livros, Games e mais Saraiva"
ex:target_www.saraiva.com.br	foaf:page	"https://www.saraiva.com.br"
ex:www.saraiva.com.br	rdf:type	foaf:Document
ex:www.saraiva.com.br	foaf:title	"Livros, Games e mais Saraiva"
ex:www.saraiva.com.br	foaf:page	"https://www.saraiva.com.br"
ex:busca.saraiva.com.br	rdf:type	foaf:Document
ex:busca.saraiva.com.br	foaf:title	"Busca Saraiva"
ex:busca.saraiva.com.br	foaf:page	"https://busca.saraiva.com.br"

Figure 10: Dados salvos no GraphDB programaticamente.

A imagem a seguir mostra resultados no frontend de uma busca no DuckDuckGo por informações do site saraiva:

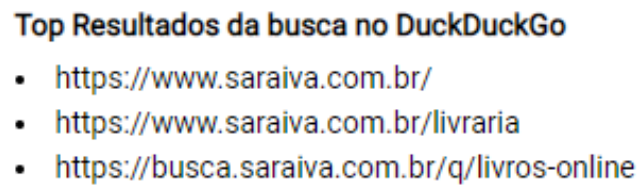
- 
- The screenshot shows the top results of a search on DuckDuckGo for the term 'Saraiva'. The results are as follows:
- Top Resultados da busca no DuckDuckGo**
- <https://www.saraiva.com.br/>
 - <https://www.saraiva.com.br/livraria>
 - <https://busca.saraiva.com.br/q/livros-online>

Figure 11: Top resultados de uma busca por informações da Saraiva no DuckDuckGo .

As imagens a seguir ilustram resultados apresentados no frontend.

RDF (TURTLE)

```
@prefix ex:<http://example.org/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

ex:target_www.saraiva.com.br a foaf:Document;
    foaf:title "Livros, Games e mais | Saraiva";
    foaf:page "https://www.saraiva.com.br" .

ex:www.saraiva.com.br a foaf:Document;
    foaf:title "Livros, Games e mais | Saraiva";
    foaf:page "https://www.saraiva.com.br" .

ex:busca.saraiva.com.br a foaf:Document;
    foaf:title "Busca | Saraiva";
    foaf:page "https://busca.saraiva.com.br" .
```

Figure 12: RDF Turtle gerado a partir dos metadados do website alvo e dos top resultados.

RDFs isomórficos

```
{
  "ex:www.saraiva.com.br": {
    "rdf:type": "foaf:Document",
    "foaf:title": "Livros, Games e mais | Saraiva",
    "foaf:page": "https://www.saraiva.com.br"
  }
}
```

Figure 13: RDF isomórficos encontrados para o site Saraiva.

7 Conclusão

Para este projeto é possível concluir algumas coisas, tantos prós como contras. Na lista de prós há que o fluxo da aplicação foi implementado em sua totalidade, ou seja, o usuário inputa um website, há um cálculo em cima disso e o resultado é oferecido de volta. Também foi possível adquirir um conhecimento sobre como utilizar a API do GraphDB. Com este projeto foram necessárias muitas buscas sobre RDF e SPARQL, portanto, ajudou a expandir o conhecimento na área.

Na lista de contras é possível ver que o conteúdo do artigo não foi implementado em sua totalidade, isto por que o artigo utiliza muitos mais atributos para identificar um website e ainda aplica um modelo de machine learning para auxiliar na classificação. Outra problemática é que a detecção não é eficiente e possui falhas, por exemplo, bloqueios pelo buscador (Google) prejudicam a avaliação, a falta de mais informações para identificar um website também prejudicam a busca por resultados, além disso sobrou alguns "hardcodings" na implementação que precisam ser resolvidos (por exemplo, prefixos SPARQL). Por fim, não foi possível encontrar um website phishing verdadeiro para se fazer uma classificação considerando um cenário real.

References

- [1] Muppavarapu, Phishing Detection using RDF and Random Forests.
(<http://iajit.org/PDF/September%202018,%20No.%205/10600.pdf>)
- [2] Código Fonte: <https://github.com/mvsantos013/trabalho-final-top-esp-bd>