

Polyana Bezerra da Costa

Matrícula: 2012409

Projeto Final de Programação: Ferramenta para anotação de *dataset* de *Fake News* em Memes

Rio de Janeiro, Brasil

Agosto - 2021

Polyana Bezerra da Costa
Matrícula: 2012409

**Projeto Final de Programação: Ferramenta para
anotação de *dataset* de *Fake News* em Memes**

Trabalho apresentado ao coordenador do Programa de Pós-Graduação em Informática da PUC-Rio como requisito para obtenção de nota na disciplina INF2102 - Projeto Final de Programação.

Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro

Orientador: Prof. Dr. Sérgio Colcher

Rio de Janeiro, Brasil
Agosto - 2021

Sumário

1	ESPECIFICAÇÃO DO PROJETO	3
1.1	Finalidade	3
1.2	Escopo	4
1.3	Requisitos	4
1.3.1	Requisitos Funcionais	4
1.3.1.1	Anotação semi-automática de faces	4
1.3.1.2	Anotação do texto através de OCR	5
1.3.1.3	Anotação Livre	5
1.3.2	Requisitos Não-Funcionais	6
1.4	Estrutura da Anotação	6
2	ARQUITETURA E PROJETO	10
2.0.1	Padrões de Projeto	11
2.0.2	Visão Geral do Sistema	11
3	TESTES	14
4	DOCUMENTAÇÃO PARA O USUÁRIO	17
	REFERÊNCIAS	23

1 Especificação do Projeto

1.1 Finalidade

Este documento apresenta uma ferramenta para anotação de uma base de imagens de *Fake News* em *memes*. Nos últimos anos, com a popularização das redes sociais, *blogs* e fóruns *online* para compartilhamento e discussão de informações, houve um aumento na disseminação de informações de conteúdo falso ou enganoso, bem como discurso de ódio (SHU; WANG; LIU, 2019). Algumas dessas "notícias" são cuidadosamente montadas para parecerem artigos em revistas, enquanto outras são apresentadas em imagens, no formato de *memes* (GUESS; NAGLER; TUCKER, 2019; SÀBAT, 2019). Com isso, ferramentas de *fact checking* e métodos para detecção de notícias falsas tem se popularizado e demonstram ser importantes para o combate à desinformação.

A fim de futuramente desenvolver uma ferramenta para detecção de *fake news* em *memes* em português, alunos do laboratório Telemídia construíram uma pequena base de imagens que apresentam informações falsas ou algum tipo de discurso de ódio no formato de *memes*. Entretanto, essa base de imagens não está anotada e por se tratar de *memes*, sua anotação envolve dados visuais e textuais. *Memes* são um tipo de conteúdo humorístico que geralmente se apresentam no formato de imagem com algum texto ou legenda escrito na imagem (SÀBAT, 2019). Na base de imagens em questão, os *memes* contêm pessoas (em geral personalidades públicas, como políticos) e algum texto associado a elas (em português), como exemplificado na Figura 1.



Figura 1 – Exemplo de um *meme* que contém informação visual e textual e pode apresentar conteúdo enganoso ou que promova discurso de ódio.

Tendo em vista esse contexto, este documento apresenta uma ferramenta para anotação de uma base de imagens de *Fake News* em *memes*. Para cada imagem, deve-se: 1 - anotar quem são as pessoas presentes (caso sejam personalidades públicas); 2 - transcrever o texto ou legenda presente na imagem; 3 - atribuir uma avaliação final à imagem: real ou falsa, caso o conteúdo presente nela apresente *fake news* ou discurso de ódio. Ao final de todas as anotações, com uma base de dados anotada, será possível usá-la em tarefas que envolvam a detecção de *Fake News* dentre outros.

1.2 Escopo

Este projeto tem como escopo o desenvolvimento de um anotador web para um base de imagens de *Fake News* em *memes*. Para isso, é importante que o anotador permita o carregamento das imagens, a visualização das mesmas, a anotação das faces e do texto presente nas imagens - ressaltando essas regiões de interesse através de *bounding boxes* e a anotação da classificação do conteúdo da imagem em real ou falso. Além disso, o anotador precisa salvar as anotações em um arquivo .json, carregar as anotações já salvas, renderizar o conteúdo das mesmas nas imagens correspondentes, permitindo a alteração de seu conteúdo ou que as mesmas sejam apagadas.

Para facilitar o processo de anotação, haverá um módulo de OCR (*Optical Character Recognition*) para detectar as regiões da imagem onde há texto, transcrevendo-os. Caso haja algum erro, o usuário precisa apenas corrigir ou confirmar se a detecção dos caracteres está correta. Além disso, para facilitar o processo de anotação das faces, haverá um módulo de localização de faces, onde dada uma imagem, serão identificadas as regiões que apresentam rostos.

Não faz parte do escopo desta ferramenta a classificação da imagem em si - se apresenta algum conteúdo falso ou não - mas sim, promover a anotação facilitada dos elementos da imagem por meio de Visão Computacional.

1.3 Requisitos

1.3.1 Requisitos Funcionais

Para especificar os requisitos funcionais que devem ser suportados pelo anotador, foram escolhidos três cenários para sua utilização:

1.3.1.1 Anotação semi-automática de faces

O primeiro cenário de utilização do anotador foi pensado para usuários que precisam detectar regiões da imagem onde há faces. Para auxiliar o usuário no processo de anotação,

usa-se um módulo de detecção de faces presente na API Vision do Google¹. Após executar a detecção de faces e visualizar se o resultado está correto, o usuário só precisa atribuir nomes às faces detectadas. Para essa anotação, a *label* será do tipo **face**.

1.3.1.2 Anotação do texto através de OCR

O segundo cenário de uso da ferramenta inclui a anotação de regiões da imagem onde há texto. Para auxiliar o usuário e evitar que este digite o texto contido em cada região, usa-se um módulo de transcrição automática do texto baseado em OCR (*Optical Character Recognition*). Com esse módulo, o usuário precisa apenas confirmar se o resultado do OCR está correto, ou editá-lo, caso não esteja. O módulo de OCR usado nesta ferramenta também está presente API Vision do Google. Usou-se o módulo *Document Text Detection*², já que além de extrair o texto de uma imagem, também fornece informações sobre blocos de textos, parágrafos, palavras e quebras de linha. Para essa anotação, a *label* será do tipo **texto**.

1.3.1.3 Anotação Livre

O terceiro cenário de uso da ferramenta permite um uso mais geral. O usuário pode selecionar regiões na imagem livremente, atribuindo a elas a *label* desejada (não precisa ser exclusivamente do tipo **face** ou **texto**).

Tendo em vista estes cenários de uso, temos os seguintes requisitos funcionais referentes à ferramenta:

- **Carregar dataset:** A ferramenta deve permitir que o usuário selecione o *dataset* de imagens que quer usar, informando o caminho em que os arquivos estão armazenados no computador do usuário;
- **Anotação semi-automática de faces:** A ferramenta deve permitir a anotação de faces através do módulo de *Face Detection*. As regiões que contém faces são detectadas pelo sistema, cabendo ao usuário apenas dar *labels* às diferentes faces;
- **Anotação semi-automática de texto:** A ferramenta deve permitir a identificação de regiões da imagem que contém texto, transcrevendo-o através de OCR. O usuário deve apenas confirmar se o resultado do OCR está correto, ou editá-lo;
- **Anotar *bounding boxes*:** A ferramenta deve permitir a anotação geral de regiões da imagem, onde o usuário seleciona a região através de uma *bounding box* e dá a ela uma *label*. Essa *label* não precisa ser dos tipos *face* ou *texto*;

¹ <https://cloud.google.com/vision>

² <https://cloud.google.com/vision/docs/handwriting>

- **Visualizar anotações:** A ferramenta deve permitir que o usuário visualize as anotações nas imagens. Para cada imagem, devem ser mostradas as *bounding boxes* e suas respectivas labels;
- **Exportar anotações:** A ferramenta deve permitir que as anotações sejam salvas no formato JSON, permitindo o download desses arquivos. Para cada imagem, a ferramenta deve associar um arquivo JSON com informações sobre a localização de cada *bounding box* e suas *labels*.

1.3.2 Requisitos Não-Funcionais

Dentre os requisitos não-funcionais que a ferramenta deve apresentar, listam-se:

- **RNF01 - Performance:** Refere-se ao tempo de resposta durante o uso das funcionalidades fornecidas pelo sistema;
- **RNF02 - Usabilidade:** Refere-se à facilidade de aprendizagem e uso do sistema;
- **RNF03 - Confiabilidade:** Deve-se buscar baixa taxa de erro durante o uso do sistema e robustez para resolvê-los;
- **RNF04 - Padrões:** Refere-se à conformidade do desenvolvimento do sistema com padrões de software em geral;
- **RNF05 - Compatibilidade:** Refere-se à compatibilidade de uso do sistema em OS Windows e Linux;

1.4 Estrutura da Anotação

As anotações feitas em cada imagem são salvas em arquivos .JSON (um arquivo para cada imagem), como é costume das ferramentas de anotação já existentes. Escolheu-se o formato JSON por este ser amplamente utilizado e facilmente interpretado em linguagens como *Python*, por exemplo. Como as anotações permitidas nesta ferramente precisam de mais informações que uma simples *label* ou *tag*, decidiu-se criar uma estrutura de anotação específica. No caso de anotações do tipo texto, além da *label* **texto**, é preciso ter a transcrição do texto detectado. Já no caso de anotações de face, além da *label* **face**, é preciso associar um nome àquela face.

Com base nisso, a estrutura de anotação usada nessa ferramenta guarda informações sobre o *dataset*, o arquivo de imagem e dos objetos presentes na imagem, guardando informações sobre a localização de cada objeto (posição da *bounding box*), a classe ao qual ele pertence e o valor desse objeto. Abaixo segue-se o detalhamento de cada item contido na estrutura do arquivo de anotação:

- *Filename*: guarda o nome do arquivo;
- *Path*: guarda o caminho completo do arquivo no sistema operacional do usuário;
- *Source*: guarda informações sobre qual o *database* ao qual a imagem pertence e a *url* do *database*, caso exista (ambos os itens podem ter valores vazios);
- *isFake*: Variável que guarda uma valor *booleano* indicando se a imagem apresenta conteúdo falso ou não;
- *Annotations*: Vetor que guarda as seguintes informações para cada região anotada:
 - type*: Guarda o tipo da anotação. Há três tipos possíveis: face, texto e geral. Anotações do tipo **geral** referem-se à objetos de qualquer outro tipo que não face e texto;
 - value*: O valor daquele tipo de anotação. Exemplo: pra uma anotação do tipo **face**, o *value* seria o nome associado à face. Para anotações do tipo **texto**, o *value* seria a transcrição do texto em si e para anotações do tipo **geral**, o *value* seria simplesmente a classe do objeto;
 - coordinates*: Posições da *bounding box* em valores inteiros para *x*, *y*, *width* e *height*.

Abaixo, segue-se um exemplo de uma anotação em imagem (Figura 2 e sua respectiva anotação num arquivo .JSON).



Figura 2 – Exemplo *bounding boxes* anotadas em uma imagem.

Listing 1.1 – Exemplo de arquivo .JSON contendo anotações da imagem.

```
[  
 {  
   "filename": "0001.jpg",  
   "path": "C:/ Downloads/Memes/",  
   "source": {
```

```
    "database": "Classic.Memes",
    "url": ""

} ,
"isFake": False,
"annotations": [
{
    "id": 001,
    "type": "face",
    "value": "Harold",
    "coordinates": {
        "x": 556,
        "y": 4,
        "width": 208,
        "height": 217
        "color": "green"
    }
},
{
    "id": 002,
    "type": "texto",
    "value": "hide\u2022the\u2022pain\u2022harold",
    "coordinates": {
        "x": 8,
        "y": 440,
        "width": 939,
        "height": 99,
        "color": "red"
    }
},
{
    "id": 003,
    "type": "geral",
    "value": "Coffee\u2022Mug",
    "coordinates": {
        "x": 565,
        "y": 293,
        "width": 101,
        "height": 134,
        "color": "blue"
    }
}
```

```
        }  
    }  
]  
}  
]
```

2 Arquitetura e Projeto

Antes de começar o desenvolvimento do anotador, era preciso determinar quais as principais necessidades e funcionalidades que o sistema deveria oferecer aos usuários. Para isso, com base nos Requisitos Funcionais apresentados na Seção 1.3.1, modelou-se um diagrama de casos de uso a fim de se ter uma visão geral do sistema. A Figura 3 apresenta os atores que utilizarão o sistema (temos o usuário e o sistema que disponibilizará as funcionalidades ao usuário) e os principais casos de uso.

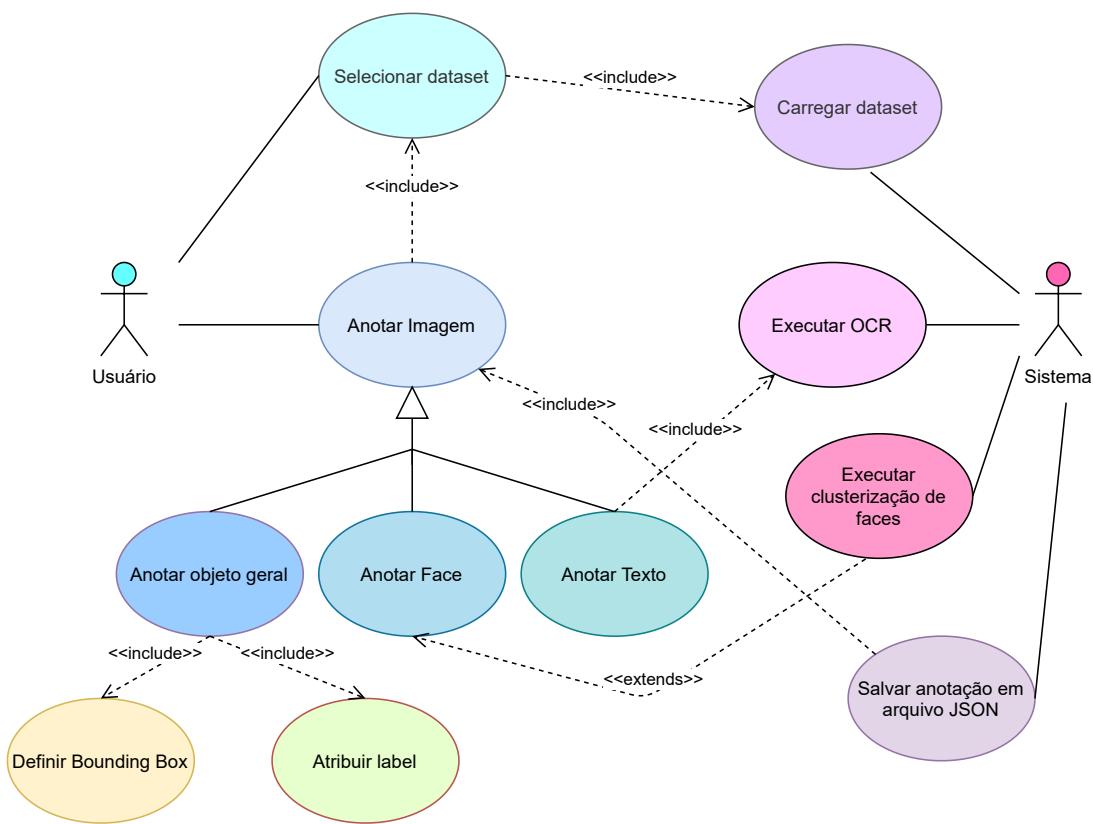


Figura 3 – Principais casos de uso da ferramenta de anotação.

Após ter as principais funcionalidades definidas, era preciso determinar quais classes comporiam o sistema, a fim de concretizar as funcionalidades desejadas. Para isso, projetou-se um pacote para conter as regras de negócio, as funcionalidades fundamentais do sistema. O pacote que contém as regras de negócio ou o modelo dessa ferramenta é chamado de pacote *Core*. As classes contidas nesse pacote são responsáveis por criar e determinar toda a estrutura de dataset de imagens e a estrutura das anotações. A Figura 4 mostra o diagrama de classes com todas as classes que compõem o pacote *Core*. O diagrama apresenta a estrutura lógica entre as classes, seus atributos e métodos e como as classes se comunicam entre si.

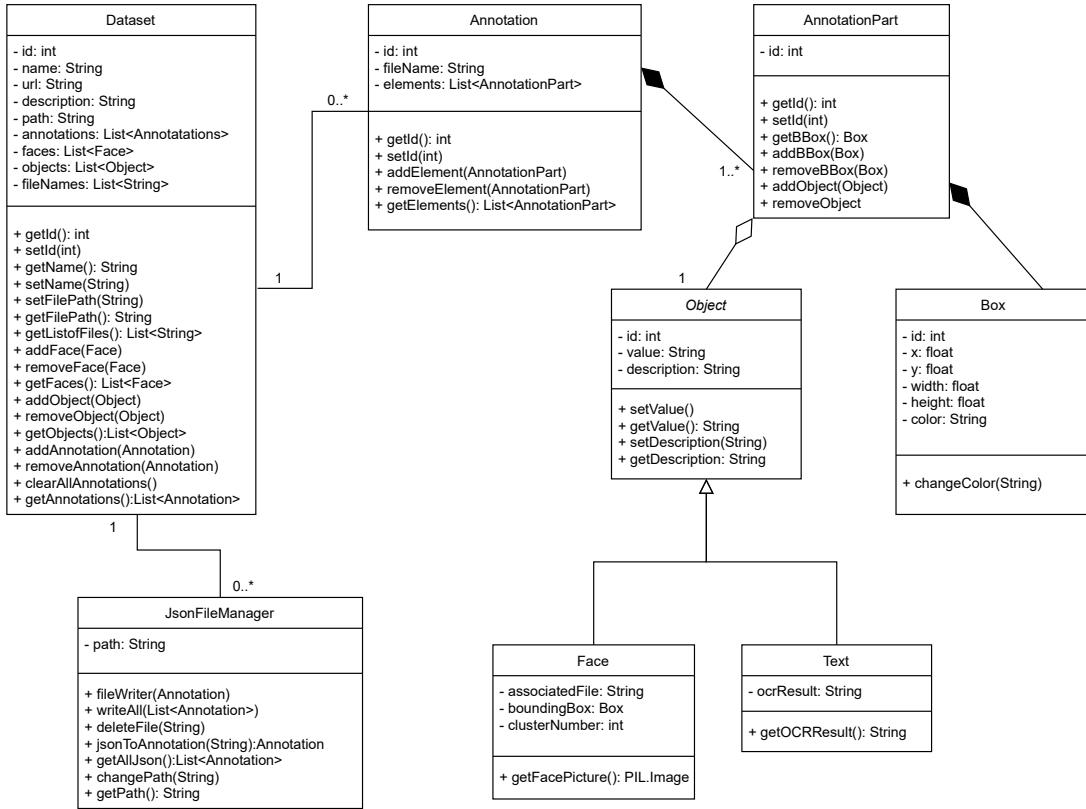


Figura 4 – Diagrama de classes da parte central do sistema, o pacote *Core*

2.0.1 Padrões de Projeto

Aplicou-se o padrão criacional **Singleton** sobre as classes *Dataset* e *JsonFileManager*, porque era desejável que essas classes tivessem apenas uma instância, a fim de manter o registro consistente das anotações que são inclusas no *Dataset* e dos arquivos json que são gerados através do *JsonFileManager*. Ao usar o padrão *Singleton*, se a classe não existir ela é criada, mas se já existir, sempre a mesma instância dela é retornada, fornecendo assim um ponto de acesso global ao objeto.

2.0.2 Visão Geral do Sistema

A ferramenta desenvolvida é uma aplicação web, que deve ser executada como um servidor local, no computador do usuário. A linguagem escolhida pro desenvolvimento da aplicação foi o *Python*, na versão 3.9.5, com o apoio da biblioteca *Dash*¹ para criar a aplicação web interativa. Para estilizar a aplicação, usou-se os componentes do toolkit para desenvolvimento front-end *Bootstrap*². Para manipulação e processamento das imagens usou-se a biblioteca *Pillow*³ e a API Vision, dentre outras.

¹ <https://dash.plotly.com/introduction>

² <https://getbootstrap.com/>

³ <https://pillow.readthedocs.io/en/stable/installation.html>

O código fonte da ferramenta se encontra no Github⁴. Todas as dependências e bibliotecas que precisam ser instaladas para o bom funcionamento da aplicação se encontram no arquivo *envname.yml* ou *requirements.txt*. Adicionalmente, para usar os módulos da API Vision, é preciso criar uma credencial ou APIKEY no site do Google Cloud. O passo a passo apresentado aqui⁵ explica como criar e ativar uma APIKEY. Depois de obter suas credenciais, copie-as e cole-as no arquivo de configuração *servicecreds.json*, que está na pasta raiz, conforme mostra a Figura 5.

```

|   callbackManager.py
|   env_name.yml
|   googleVisionAPI.py
|   imports.py
|   layoutManager.py
|   mainController.py
|   README.md
|   requirements.txt
|   run.py
|   service_creds.json
|   unitTesting.py
|   unitTestingData.py
|
+---core
|       annotation.py
|       annotationPart.py
|       box.py
|       dataset.py
|       face.py
|       jsonFileManager.py
|       object.py
|       text.py
|       __init__.py
|
+---images
|       0cmklxckncv11.jpg
|
+---jsonAnnotation
|       0cmklxckncv11.json
|
\---upload
    \---media
        \---images
            0cmklxckncv11.jpg

```

Figura 5 – Estrutura das pastas no projeto.

A Figura 5 mostra a organização das pastas no projeto. A começar pela pasta **core**, ela contém todas as classes apresentadas no diagrama de classes anteriormente,

⁴ <https://github.com/poleana/PoliticalMemesAnnotator>

⁵ <https://cloud.google.com/vision>

representando a camada de domínio ou o modelo da aplicação. A pasta raiz contém todos os arquivos usados na camada de visão da aplicação e na camada de controle, como é mostrado em mais detalhes na Figura 6. Além disso, a pasta raiz contém arquivos de configuração como `servicecreds.json`, `envname.yml`, `requirements.txt` e `imports.py` que é usado pelas classes `callbackManager.py` e `layoutManager.py`. A pasta raiz também contém os dois arquivos usados para fazer testes unitários, `unitTesting.py` e `unitTestingData.py`. Por fim, a pasta raiz tem o arquivo `run.py` que é usado para executar a aplicação. Depois de configurar o ambiente, caso queira executar a aplicação, basta digitar `python run.py` no seu prompt.

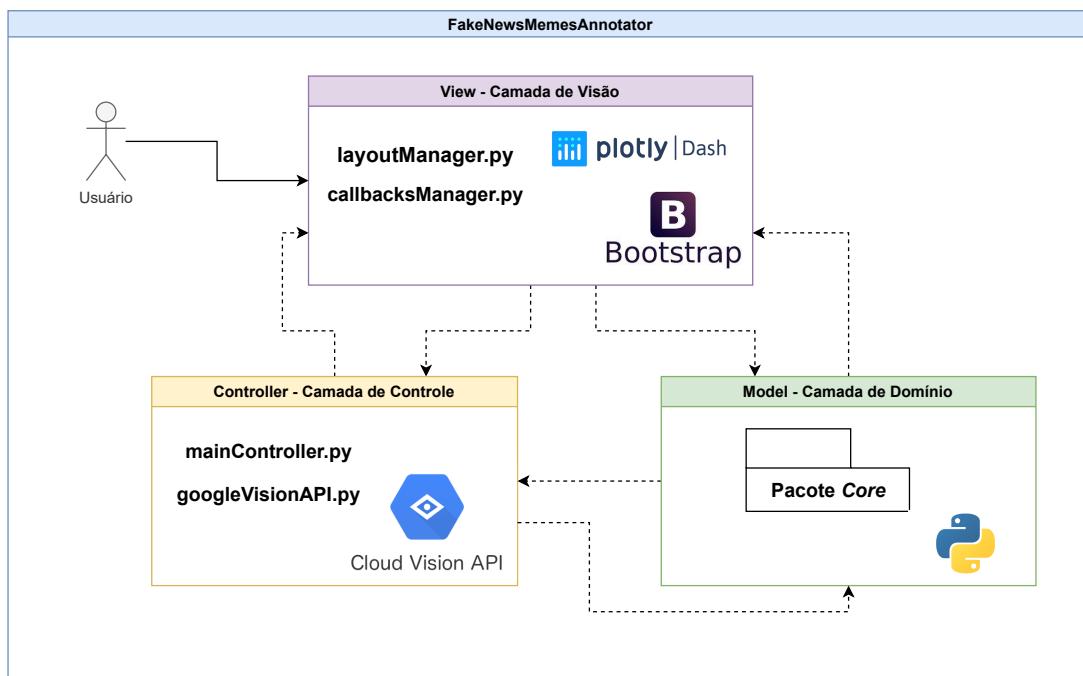


Figura 6 – Camadas da ferramenta desenvolvida.

A pasta **images** contém algumas imagens de exemplo da base de dados de *Fake News* em memes. Ao testar o programa, seria prudente usar as imagens contidas nessa pasta. No repositório do Github, essa pasta contém mais imagens de exemplo. A pasta **jsonAnnotation** contém os arquivos .json correspondentes às anotações salvas. Entretanto, na aplicação é possível fazer o download dos arquivos e então eles ficam salvos na pasta de **Downloads**. Por fim, ao fazer o *upload* de imagens para usar na aplicação, elas ficam armazenadas na pasta **upload media images**.

3 Testes

A fim de controlar e avaliar a qualidade do código desenvolvido, testes unitários foram aplicados sobre algumas das principais funcionalidades da ferramenta. Para realizar os testes de forma automatizada, usou-se a biblioteca `unittest`¹ do *Python*. Os arquivos que contém os testes e os dados usados nele são: *unitTesting.py* e *unitTestingData.py*, na pasta raiz do projeto.

Abaixo segue a descrição de cada funcionalidade testada:

- **Teste de criação de um Dataset**

Esse é o teste mais simples, que avalia apenas se uma instância da classe Dataset foi criada corretamente, com o mesmo nome que o usuário informou. O arquivo *unitTestingData.py* fornece os dados necessários para a criação do objeto Dataset, enquanto o primeiro método do arquivo *unitTesting.py* avalia se a instância foi criada corretamente. A Figura 7 mostra o código referente à classe *unitTesting.py*, e a Figura 8 mostra o resultado dos testes.

- **Teste de execução do OCR**

O módulo de OCR da API Vision também foi testado, para saber se o módulo retornava algum resultado e se os resultados eram corretos. Escolheu-se uma imagem específica e aplicou-se o módulo de OCR, selecionando-se uma *bounding box* do seu resultado. Comparou-se o resultado obtido pelo OCR com o resultado esperado para aquela região (A transcrição do texto corretamente). O teste apresentou resultados corretos, como mostram as Figuras 7 e 8. Entretanto, vale ressaltar que nesse módulo, não é mandatório que o resultado do OCR esteja totalmente correto, porque o usuário pode corrigi-lo. O importante é que módulo retorne *bounding boxes* corretas.

- **Teste de execução do Face Detector**

De forma similar ao teste anterior, o módulo de detecção de face também foi testado. Nesse teste, se queria saber se se a coordenada x de uma *bounding box* que contém uma face era um número inteiro. Com esse teste, foi possível saber se a execução desse módulo retornada resultados coerentes e se o resultado estava sendo estruturada da maneira esperada (Um vetor com 4 números inteiros representando as posições x, y, largura e altura). O teste também apresentou resultados corretos, conforme mostra a Figura 8.

- **Teste de criação de arquivo JSON**

¹ <https://docs.python.org/3/library/unittest.html>

Salvar anotações em um arquivo .json é uma das principais funcionalidades dessa ferramenta, portanto, ela também foi testada. Inicialmente, criou-se um objeto do tipo *Annotation* e um objeto do tipo *jsonFileManager* que recebeu essa anotação como parâmetro no método para geraçao de json. Para avaliar se o json foi mesmo gerado, recuperou-se o nome do arquivo gerado e verificou-se se ele existia mesmo no diretório especificado. Como mostra a Figura 8, esse teste também apresentou resultados corretos, o que mostra que os arquivos json estavam de fato sendo salvos em disco.

- **Teste de conversão do arquivo JSON em Objeto**

Por fim, avaliou-se a funcionalidade de ler um arquivo json e convertê-lo para um objeto do tipo *Annotation*. Conforme mostram as Figuras 7 e 8, o objeto foi criado corretamente, com as mesmas informações contidas no objeto *Annotation* original usado para escrita do arquivo json.

```
import unittest
from unitTestingData import *

class Tests(unittest.TestCase):

    #Testando a criação de um dataset
    def test_DatasetCreation(self):
        database = getTest_DatasetCreationData()
        self.assertEqual(database.name, "Political Memes", "A resposta deve ser Political Memes.")

    #Testando se o resultado do OCR da API Vision está correto
    def test_OCRResult(self):
        strOCR = getTest_OCRResultData()
        print("Resultado OCR: ", strOCR)
        self.assertEqual(strOCR, "BOULOS , SEM TETO MAS COM JATINHO ", "A resposta deve ser BOULOS , SEM TETO MAS COM JATINHO .")

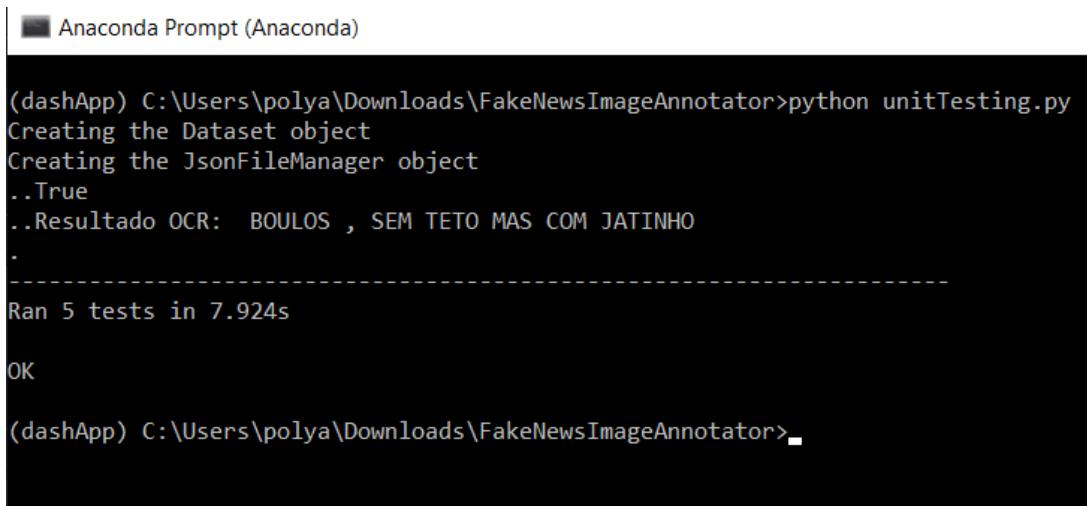
    #Testando se a posição x da bounding box da face retornada pela API Vision é um inteiro (se estiver tudo correto, deve ser)
    def test_FaceDetection(self):
        posX = getTest_FaceDetectionData()
        print(isinstance(posX, int))
        self.assertTrue(isinstance(posX, int), "A resposta deveria ser True.")

    #Testando se o arquivo json de uma anotação é criado
    def test_CreateJson(self):
        jsonName, jsonFileName = getTest_CreateJson()
        self.assertEqual(jsonName, jsonFileName, "Os arquivos deveriam ter o mesmo nome.")

    #Testando se a conversão do conteúdo de um arquivo json em anotacao está correto
    def test_JsontoAnnotation(self):
        annotation1, annotation2 = getTest_JsonToAnnotationData()
        self.assertEqual(annotation1.fileName, annotation2.fileName, "Os objetos devem corresponder a mesma imagem.")

if __name__ == '__main__':
    unittest.main()
```

Figura 7 – Funcionalidades avaliadas por meio de testes unitários.



The screenshot shows a terminal window titled "Anaconda Prompt (Anaconda)". The command entered is "python unitTesting.py". The output indicates the creation of Dataset and JsonFileManager objects, a True result for OCR, and a test result for a specific image. The test summary shows 5 tests ran in 7.924s with an OK status.

```
(dashApp) C:\Users\polya\Downloads\FakeNewsImageAnnotator>python unitTesting.py
Creating the Dataset object
Creating the JsonFileManager object
..True
..Resultado OCR: BOULOS , SEM TETO MAS COM JATINHO
.
-----
Ran 5 tests in 7.924s
OK
(dashApp) C:\Users\polya\Downloads\FakeNewsImageAnnotator>
```

Figura 8 – Resultado da execução dos testes. Pode-se ver que todas as funcionalidades passaram no teste, nenhuma delas apresentou erro.

4 Documentação para o Usuário

Após configurar o ambiente e executar o arquivo *run.py*, a aplicação deve ser carregada no endereço *http://localhost:8052/*. A tela inicial apresenta um botão intitulado **Upload de Imagens**. Ao clicar nele, aparece um outro botão para escolher arquivos para fazer *upload*. Clique em **Escolher arquivos** e selecione algumas das imagens que estão na pasta **images**. A Figura 9 ilustra esse processo.

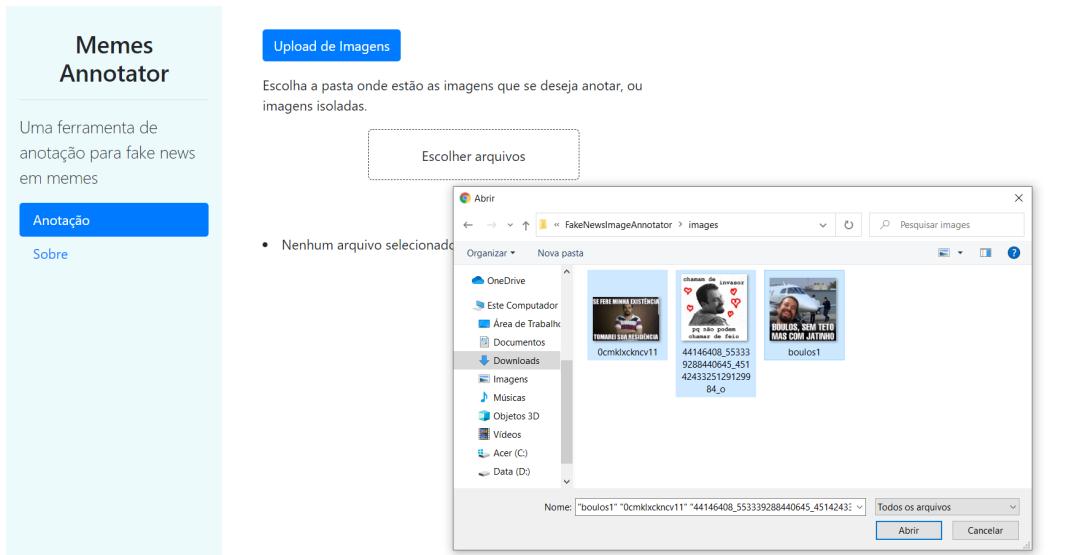


Figura 9 – Tela inicial do sistema: Fazendo upload de imagens.

Depois de concluir o upload, a primeira imagem da lista de uploads é renderizada num canvas, conforme mostra a Figura 10. O canvas permite desenhar na imagem (botão que tem símbolo de um quadrado), apagar o último elemento desenhado (seta para a esquerda), redesenhar o último elemento apagado (seta para a direita) e selecionar um objeto e movê-lo (botão com símbolo de etiqueta). Abaixo do canvas aparecem alguns botões que controlam qual imagem está sendo renderizada no momento, para avançar ou retroceder na lista de imagens, clique no botão correspondente. O botão **atualizar** é usado quando atualizar o conteúdo da imagem quando os módulos de OCR, detecção de face e edição de anotação estão em uso.

Para desenhar *bounding boxes* na imagem, clique no botão com símbolo de retângulo (*Rectangle Tool*, escolha a cor da linha, se desenhar, e em seguida, desenhe um retângulo na região desejada. Cada *bounding box* deve ser adicionada por vez. Após desenhar o retângulo na área desejada, preencha os campos **Tag** com um rótulo para aquela *bounding box* e selecione o tipo de anotação. Na Figura 11, vê-se que foi anotada uma região do tipo face com o rótulo "Guilherme". Para adicionar essa parte de anotação, clique no botão



Figura 10 – Canvas para desenho na imagem.

Adicionar Forma. Adicione e anote uma *bounding box* por vez. Somente clique no botão **Salvar Anotação** quando tiver terminado suas anotações.

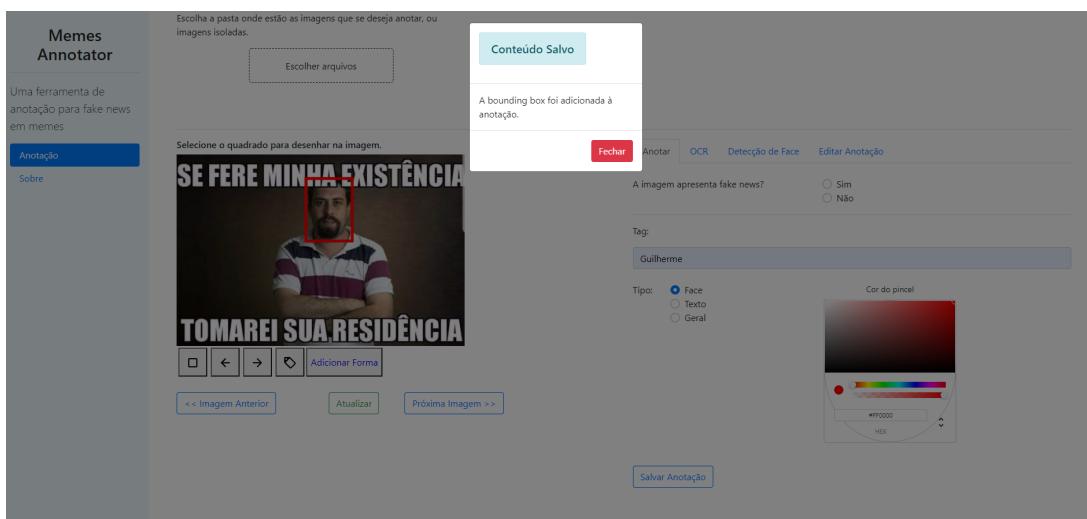


Figura 11 – Adicionando uma *bounding box* à anotação.

A guia **Anotar** representa as anotações livres, onde o usuário desenha as *bounding*

boxes e informa as tags. Também é possível usar a guia **OCR** para executar o módulo de OCR e identificar textos automaticamente. Ao clicar no botão **Gerar OCR**, uma dropdown com a lista de *bounding boxes* resultantes é mostrada, conforme ilustra a Figura 12. Ao selecionar uma opção da lista, a *bounding box* correspondente é desenhada na imagem, e o texto obtido através do OCR é mostrado no componente **Tag**. Na Figura 13 podemos ver que o OCR detectou o texto na região delimitada por uma caixa vermelha como "Tomarei sua residência".

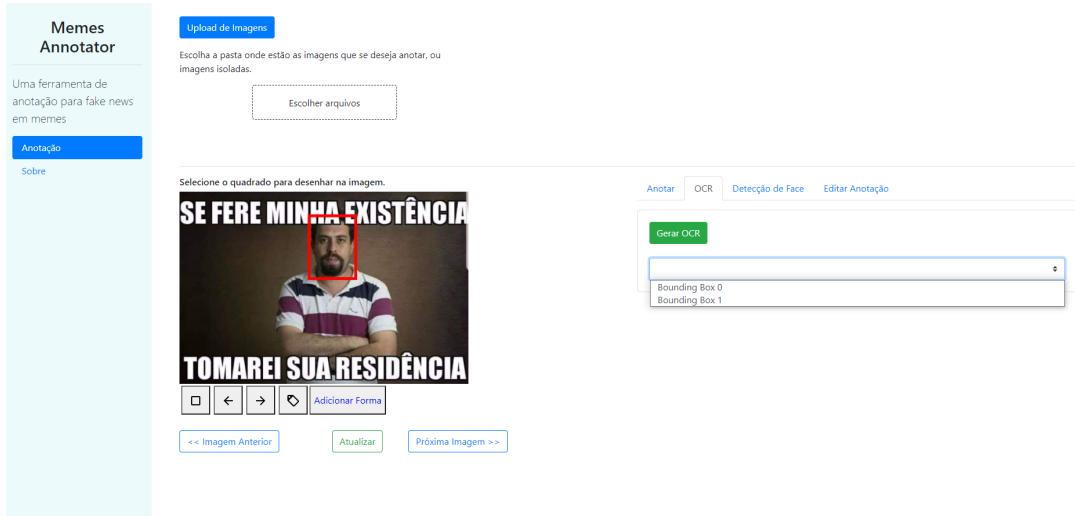


Figura 12 – Lista de *bounding boxes* obtidas com o OCR.

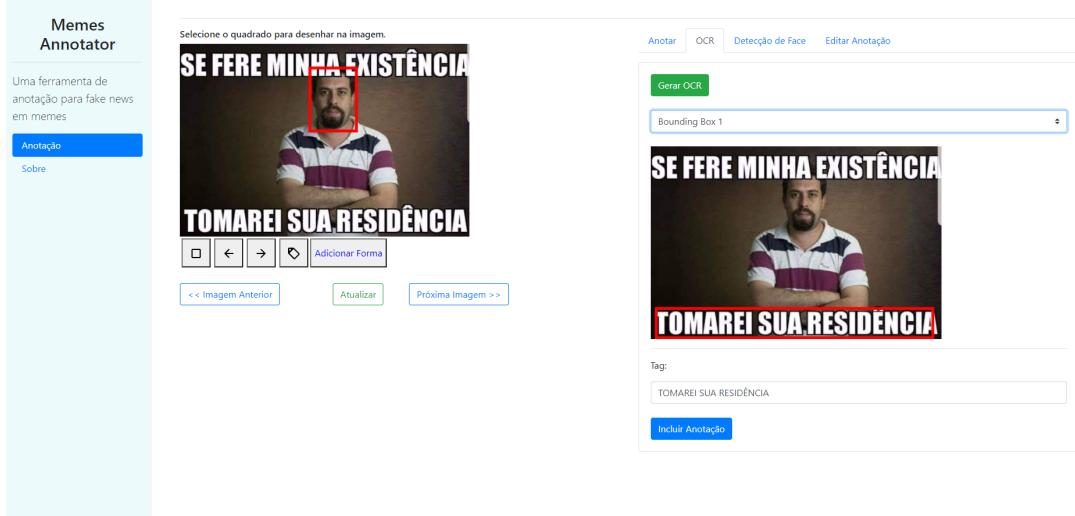


Figura 13 – Resultado do OCR para a *bounding box* selecionada na tela anterior.

Se achar que o resultado do OCR está correto e deseja adicioná-lo à sua anotação, clique no botão **Incluir Anotação**. Se o texto transscrito estiver incorreto, você pode corrigir o valor do input antes de salvar a anotação. Depois de adicionar uma anotação vinda do módulo de OCR, é importante clicar no botão **Atualizar**, para que a nova região seja desenhada na imagem.

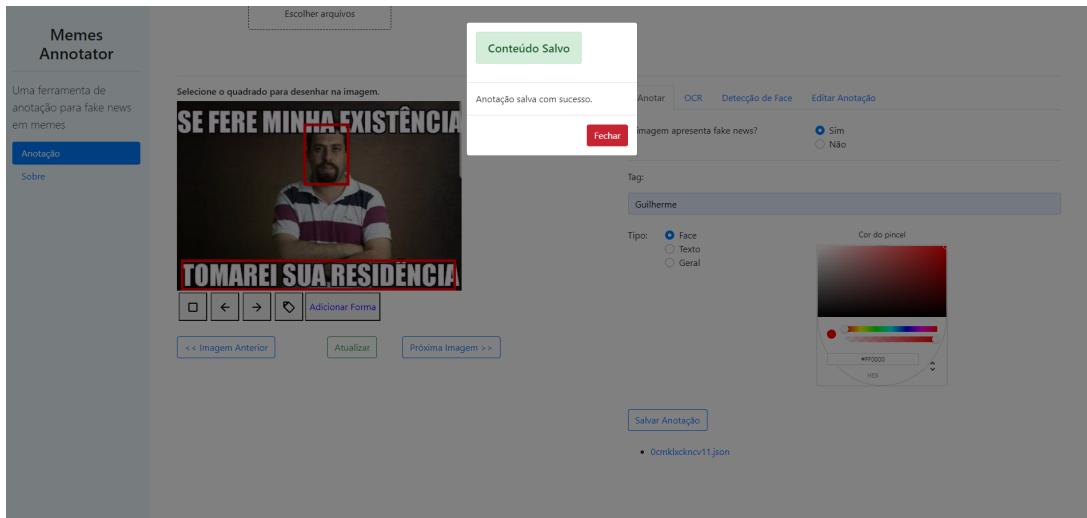


Figura 14 – Anotação Salva. Note que abaixo do botão de salvar anotação aparece o link para download do arquivo .json gerado.

Caso tenha terminado de adicionar *bounding boxes*, pode salvar a anotação, se quiser. Antes de aperta o botão **Salvar Anotação**, preencha Sim ou Não para indicar se a imagem apresenta conteúdo falso. Se não clicar numa dessas opções, a anotação não será salva. Conforme mostra a Figura 14, após salvar a anotação, aparece um popup informando se a anotação foi mesmo salva e um link para baixar o arquivo .json aparece logo abaixo do botão de salvar anotação.

Uma outra guia presente no anotador é a guia de **Detecção de Face**, que funciona de forma analoga à guia de OCR. Ao clicar no botão **Localizar Faces**, uma dropdown com uma lista de faces detectadas é mostrada. Selecione uma opção e veja o resultado da detecção mostrado na imagem.



Figura 15 – Resultado da detecção de faces.

Como pode-se ver na Figura 15, uma *bounding box* com a área referente a face

é desenhada na imagem. Caso o usuário queira incluir essa região na anotação, basta preencher o campo **Tag** e apertar o botão **Incluir Anotação**. Em seguida, aperte o botão **Atualizar** para que a nova *bounding box* seja desenhada na imagem renderizada no canvas.

A última guia **Editar Anotação** permite a edição e apagamento de partes de anotações, ou *bounding boxes* já inclusas na imagem. Ao clicar no botão **Selecionar Bounding Box**, uma lista com todas as regiões já anotadas na imagem será mostrada. Conforme ilustra a Figura 16, ao selecionar uma opção, somente aquela região será desenhada na imagem. É possível editar a tag daquela anotação e o tipo também. Caso faça modificações, aperte no botão **Salvar edição** para salvá-las; em seguida, aperte o botão **Atualizar**.

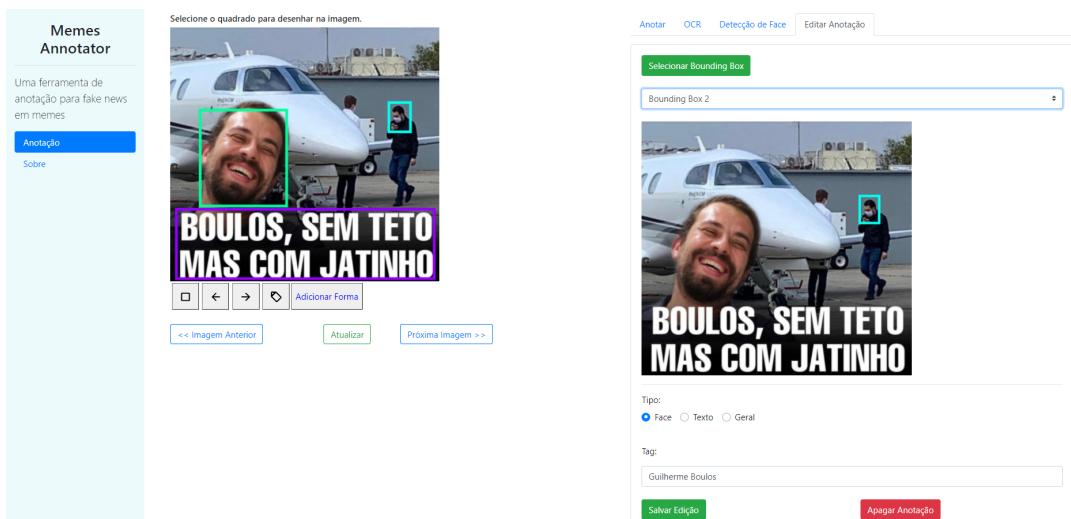


Figura 16 – Tela de edição de anotações.

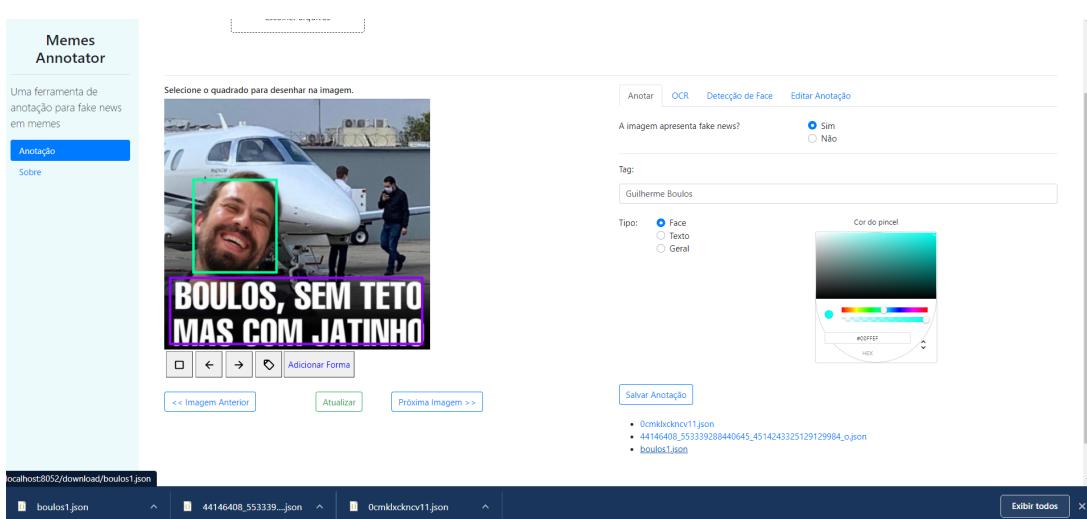


Figura 17 – Imagem atualizada depois de apagar uma das bounding boxes.

Sempre que modificar o conteúdo da anotação, adicionando ou retirando dados, é preciso apertar o botão **Salvar Anotação** novamente (caso ainda não o tenha feito),

para atualizar o arquivo json correspondente à anotação. Na Figura 17, é possível ver que o usuário escolheu apagar uma das *bounding boxes*, tanto que a imagem renderizada no canvas mostra apenas dois retângulos desenhados e não três, como anteriormente.

Por fim, após terminar as edições, ao clicar no botão **Salvar Anotação**, e baixar o arquivo json correspondente à anotação, pode-se ver o conteúdo da anotação salvo no arquivo e coerente com aquilo que foi inserido pela interface, conforme ilustra a Figura 18.



```

boulos1.json ×
C: > Users > polya > Downloads > boulos1.json > ...
1  {
2    "filename": "boulos1.png",
3    "path": "c:\\users\\polya\\Downloads\\FakeNewsImageAnnotator\\upload\\media\\images\\",
4    "source": {"database": "Political Memes", "url": ""},
5    "isFake": [false],
6    "annotations": [
7      {
8        "id": 0, "type": "face", "value": "Guilherme Boulos",
9        "coordinates": {"x": 67, "y": 191, "width": 204, "height": 226, "color": "#00ffffa"}
10      },
11      {
12        "id": 1, "type": "texto", "value": "Boulos, sem teto mas com jatinho",
13        "coordinates": {"x": 11, "y": 425, "width": 609, "height": 163, "color": "#9400ff"}
14      }
15    ]
16  }

```

Figura 18 – Arquivo JSON resultante da anotação.

As principais funcionalidades da aplicação foram apresentadas nesta seção e o código presente no GitHub¹ está bem comentado. Entretanto, caso tenha alguma dúvida, entre em contato pelo *e-mail*: polyanabcosta@gmail.com.

¹ <https://github.com/poleana/PoliticalMemesAnnotator>

Referências

GUESS, A.; NAGLER, J.; TUCKER, J. Less than you think: Prevalence and predictors of fake news dissemination on facebook. *Science advances*, American Association for the Advancement of Science, v. 5, n. 1, p. eaau4586, 2019. Citado na página 3.

SÀBAT, B. O. *Multimodal Hate Speech Detection in Memes*. Dissertação (B.S. thesis) — Universitat Politècnica de Catalunya, 2019. Citado na página 3.

SHU, K.; WANG, S.; LIU, H. Beyond news contents: The role of social context for fake news detection. In: *Proceedings of the twelfth ACM international conference on web search and data mining*. [S.l.: s.n.], 2019. p. 312–320. Citado na página 3.