Guilherme Morone Araujo

Detecção Automática de Falhas de Localização a partir de Imagens

Universidade Federal de Pernambuco
dcampelo@cin.ufpe.br
https://www.cin.ufpe.br/~tg

Recife
2024

Guilherme Morone Araujo

Detecção Automática de Falhas de Localização a partir de Imagens

Trabalho apresentado ao Programa de Graduação em Engenharia da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Graduado em Engenharia da Computação.

Área de Concentração: *Engenharia de Software*
Orientador: *Breno Alexandro Ferreira de Miranda*

Recife
2024

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Araujo, Guilherme Morone.
Detecção automática de falhas de localização a partir de imagens / Guilherme Morone Araujo. - Recife, 2024.
46 p : il., tab.

Orientador(a): Breno Alexandro Ferreira de Miranda
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Informática, Engenharia da Computação - Bacharelado, 2024.
Inclui referências, apêndices.

1. Desenvolvimento Web. 2. Localização e Internacionalização. 3. OCR. 4. Pesquisa e Usabilidade. 5. Python e React. I. Miranda, Breno Alexandro Ferreira de. (Orientação). II. Título.

000 CDD (22.ed.)

GUILHERME MORONE ARAUJO

Detecção Automática de Falhas de Localização a partir de Imagens

Trabalho apresentado ao Programa de Graduação em Engenharia da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Graduado em Engenharia da Computação. Área de concentração: Engenharia de Software

Aprovado em: 15/08/2024.

BANCA EXAMINADORA

_____

Prof. Dr. Breno Alexandro Ferreira de Miranda (Orientador)
Universidade Federal de Pernambuco - UFPE

_____

Prof. Dr. Kiev Santos da Gama (Examinador Interno)
Universidade Federal de Pernambuco - UFPE

Dedico esta dissertação a toda minha família e amigos.

RESUMO

A Localização (L10N) e Internacionalização (I18N) são processos importantes para a acessibilidade de aplicativos a todas as pessoas. Porém, muitas ferramentas acabam produzindo erros que podem ser difíceis de serem detectados, ou que necessitem de um trabalho manual extenso para tal. O objetivo deste trabalho é fornecer um suporte ferramental para auxiliar desenvolvedores e testadores na identificação automática de problemas de Localização (L10N), de modo a construir uma aplicação web para detectar, em particular, problemas de Localização (L10N) relacionados a ausência de tradução ou tradução incorreta. Com o objetivo de avaliar a solução proposta, um estudo foi realizado para investigar os critérios ideais para a construção da ferramenta. Com isso, um Questionário de Usabilidade (SUS) foi adotado. Os resultados obtidos a partir deste questionário indicam que a pontuação final foi de **93,33**, sendo a classificação do trabalho apresentado como Aceitável (Nota A - Excelente).

Palavras-chave: Localização, Internacionalização, Captura de Tela, Imagem, OCR, Python, Desenvolvimento Web.

ABSTRACT

Localization (L10N) and Internationalization (I18N) are important processes for the accessibility of applications to all people. However, many tools end up producing errors that can be difficult to detect, or that require extensive manual work to do so. The objective of this work is to provide a tool support to assist developers and testers in the automatic identification of Localization (L10N) problems, in order to build a web application to detect, in particular, Localization (L10N) problems related to lack of translation or incorrect translation. In order to evaluate the proposed solution, a study was carried out to investigate the ideal criteria for building the tool. Therefore, a Usability Questionnaire (SUS) was adopted. The results obtained from this questionnaire indicate that the final score was **93.33**, with the classification of the work presented as Acceptable (Grade A - Excellent).

Keywords: Localization, Internationalization, Screenshot, Image, OCR, Python, Web Development.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

GCP	*Google Cloud Platform*
I18N	Internacionalização
L10N	Localização
OCR	Reconhecimento Óptico de Caracteres
SUS	Questionário de Usabilidade
XML	*Extensible Markup Language*

SUMÁRIO

1

INTRODUÇÃO

Com os mercados globais em expansão, empresas de *software* buscam ampliar suas vendas em um mercado internacional crescente para aplicações multilíngues. Nesse contexto, a Internacionalização (I18N) e a Localização (L10N) surgiram como processos cruciais para atingir uma audiência global e garantir que o produto seja eficiente e bem recebido em diferentes mercados ao redor do mundo (1). A I18N prepara um *software*, produto, aplicativo ou conteúdo para suportar diferentes línguas e culturas, abstraindo elementos específicos de idioma e cultura do código (1). Já a L10N adapta o *software*, produto, aplicativo ou conteúdo internacionalizado para mercado ou região específicos, ajustando-o conforme as peculiaridades culturais, legais e técnicas da região alvo (1).

Apesar dos contínuos avanços, ainda é um desafio garantir que o *software* se adapte perfeitamente às particularidades de cada localidade, incluindo língua, cultura e leis, o que pode requerer mudanças significativas na aparência, funcionalidade e apresentação dos dados do *software*. Frequentemente, desenvolvedores não possuem profundo conhecimento das línguas e culturas dos mercados-alvo, destacando a importância de ferramentas especializadas e a necessidade de colaboração com especialistas locais (1).

Por este motivo, é crucial buscar alternativas que auxiliem no desenvolvimento de soluções para L10N, melhorando o processo e garantindo um resultado preciso na adaptação às peculiaridades de cada localidade, o que pode gerar economia de recursos durante o desenvolvimento e assegurar que o *software* atenda às expectativas culturais, linguísticas e legais específicas de cada mercado-alvo (2).

O objetivo deste trabalho é fornecer um suporte ferramental para auxiliar desenvolvedores e testadores na identificação automática de problemas de L10N, de modo a construir uma aplicação web para detectar, em particular, problemas de L10N relacionados a ausência de tradução ou tradução incorreta.

A estrutura do restante do documento está organizado da seguinte forma:

1. **Revisão da Literatura**: Onde é abordado como a literatura e trabalhos anteriores auxiliam este estudo.

2. **Metodologia**: Onde é apresentado mais informações sobre a metodologia do trabalho,

coleta e análise dos dados.

3. **Ferramenta**: Onde é apresentado como a ferramenta foi construída em detalhes.
4. **Avaliação**: Onde são apresentados os resultados do estudo e avaliações.
5. **Conclusão**: Onde são apresentadas as conclusões do estudo e recomendações para estudos futuros.

2

REVISÃO DA LITERATURA

A revisão da literatura foi realizada a partir do Google Scholar (3) usando os termos "detecção"(*detection*), "falhas"(*errors*), "localização"(*localization*) e "l10n", com filtro ativo para pesquisa publicadas nos últimos 5 anos em inglês ou português.

2.1 TRABALHOS RELACIONADOS

2.1.1 Falhas de Localização

Segundo Zalieskaitė (4), muitos aplicativos e sites realizam o processo de L10N de forma manual, gerando muitos erros. O processo de L10N deve ser feito levando em conta diferentes aspectos, como: formatação de tempo, unidades de medidas e moedas; nomes de arquivos; alfabeto; abreviaturas; dentre outros. Por isso, a criação de ferramentas para automatizar e facilitar a detecção destes erros é de extrema relevância.

Há diversos tipos de erros de L10N, alguns deles serão listados adiante. Porém, o foco deste trabalho é na falha de Ausência de Tradução ou Tradução Incorreta.

2.1.1.1 Ausência de Tradução ou Tradução Incorreta

Ocorre quando o texto não é traduzido para o idioma alvo corretamente. Em alguns casos, a palavra ou texto não são traduzidos por completo. A figura 1 mostra um exemplo de ausência de tradução.

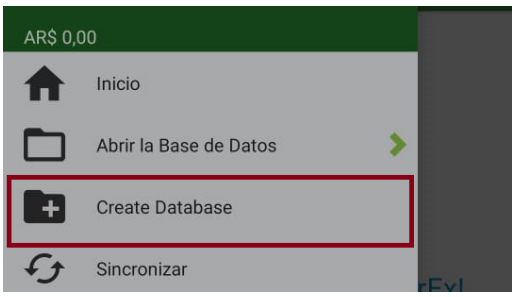

Figura 1: Exemplo de Ausência de Tradução ou Tradução Incorreta

2.1.1.2 *Elipse*

A elipse se dá quando a palavra após a tradução fica muito longa e o espaço não é suficiente, resultando em um corte com reticências ("..."), que pode ocultar informações relevantes ou gerar palavras grosseiras. A figura 2 mostra um exemplo de elipse.

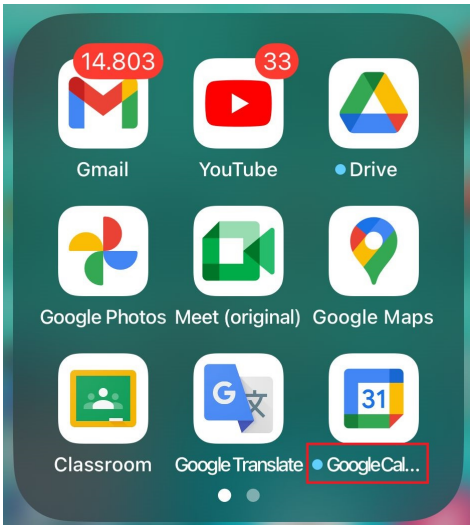

Figura 2: Exemplo de Elipse

2.1.1.3 *Truncamento*

Pode ser confundido com a elipse, porém, neste caso, o corte não é simbolizado com reticências. A palavra é cortada de forma repentina. A figura 3 mostra um exemplo de truncamento.

Figura 3: Exemplo de Truncamento

2.1.1.4 *Sobreposição*

Acontece quando o texto é sobreposto por elementos gráficos, como um *modal* (elemento *web* que aparece a frente de todos os outros e desabilita o resto do conteúdo da página), dificultando a legibilidade. A figura 4 mostra um exemplo de sobreposição.

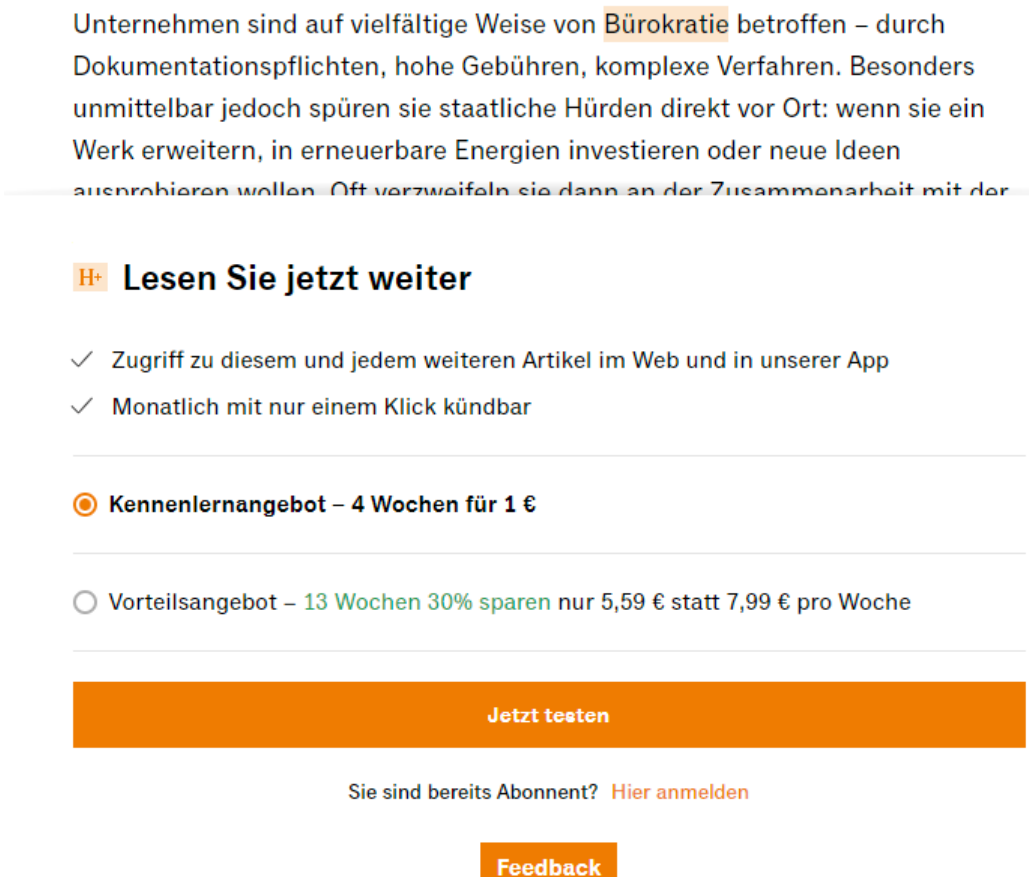

Figura 4: Exemplo de Sobreposição

2.1.2 Ferramenta para Detecção Automática de Falhas de Localização

Lucena (5) desenvolveu um estudo sobre uma ferramenta para detectar falhas de L10N em aplicativos Android, convertendo capturas de tela em *Extensible Markup Language* (XML) e gerando um relatório de erros e que serviu de inspiração para o presente estudo.

Porém, essa ferramenta fica limitada a aplicativos móveis com sistema operacional Android, além de não possuir uma maneira mais fácil de executar (é executada diretamente via um arquivo Python, com vários parâmetros). Portanto, a principal experimentação e inspiração para atingir o objetivo final deste trabalho será aprimorar a solução proposta pelo autor, de modo a construir uma alternativa que seja possível detectar erros em qualquer imagem e que esteja disponível na *web*.

2.1.3 Outros Estudos no Contexto de Localização

Outros trabalhos auxiliam neste estudo, entre eles:

- Pesquisa que visa avaliar a utilidade da ferramenta proposta, que auxilia no treinamento de testadores de I18N e L10N (6).

- Pesquisa que visa propor e avaliar soluções automatizadas para apoiar testes de I18N e L10N (7).
- Pesquisa que abordar e resolver os desafios de L10N e I18N através de uma ferramenta sofisticada, chamada TString, além de avaliar sua importância para a literatura (8).
- Pesquisa que introduz uma técnica automatizada para detectar problemas de I18N e L10N em aplicações *web* (9).

Todas essas pesquisas se relacionam com o presente trabalho para mostrar alternativas já introduzidas na literatura acerca de detecção de falhas de L10N.

2.1.4 Critérios de Usabilidade

Para a construção de uma ferramenta adequada, os critérios de usabilidade devem ser investigados e seguidos. Silva (10) cita a importância da qualidade de *software*, através da demonstração das prioridades e abordagens a serem seguidas para satisfazer as necessidades do usuário. A fim de criar um sistema com alta qualidade de *software*, um Questionário de Usabilidade (SUS) foi realizado com base em estudos feitos por Brooke (11) e Bangor *et al.* (12), que avaliam o sistema baseado em perguntas subjetivas.

3

METODOLOGIA

A fim de alcançar o objetivo do trabalho, três pontos importantes foram levados em conta:

- Garantir a facilidade de uso e manutenção da ferramenta, além da eficiência e confiabilidade (imunidade a falhas).
- Utilizar bibliotecas e outras ferramentas atualizadas e de alto nível, para melhor escalabilidade do projeto.
- Buscar opiniões de outras pessoas da área e fazer pesquisas acerca dos melhores critérios de usabilidade.

Para isso, a principal fonte de dados da ferramenta é o conjunto de imagens enviado pelo usuário na plataforma. Como será explicado mais adiante, o usuário poderá também fazer o *upload* de outros arquivos e configurações, como dicionários customizados. Essas informações serão utilizadas apenas para processar e detectar possíveis erros e não serão armazenadas remotamente, apenas de forma local, sendo disponíveis apenas no navegador do usuário.

Já para buscar opiniões de outras pessoas da área e fazer pesquisas acerca dos melhores critérios de usabilidade, um SUS *online* foi disponibilizado para pessoas da área. Com base no estudo de Brooke (11) em 1995, o questionário foi criado com 10 perguntas diferentes e disponibilizado para pessoas do Centro de Informática (CIn - UFPE) através de um formulário on-line. Apenas a pergunta 8 foi alterada em relação ao estudo de Brooke, todas as outras mantiveram o mesmo enunciado, pois a oitava pergunta original era a negação da terceira pergunta e, por isso, poderia parecer duplicada.

As perguntas são mostradas a seguir e o usuário deve marcar o quão fortemente ele concorda com cada afirmação, variando de "Discordo Totalmente"até "Concordo Totalmente"(12) em uma escala de Likert (13) de cinco pontos, ou seja, o valor de cada item varia de 1 a 5. É importante ressaltar que este formulário mantém o anonimato.

1. Acho que gostaria de utilizar esta ferramenta com frequência.
2. Considerei a ferramenta mais complexa do que o necessário.

3. Achei a ferramenta fácil de utilizar.
4. Acho que necessitaria de ajuda de um técnico para conseguir utilizar esta ferramenta.
5. Considerei que as várias funcionalidades desta ferramenta estavam bem integradas.
6. Achei que esta ferramenta tinha muitas inconsistências.
7. Suponho que a maioria das pessoas aprenderia a utilizar rapidamente esta ferramenta.
8. Achei esta ferramenta ineficiente.
9. Senti-me muito confiante ao utilizar esta ferramenta.
10. Tive que aprender muito antes de conseguir lidar com esta ferramenta.

4

FERRAMENTA

Nesta seção, é explicado em detalhes o funcionamento e construção da ferramenta. O fluxograma 5 detalha bem o funcionamento da aplicação.

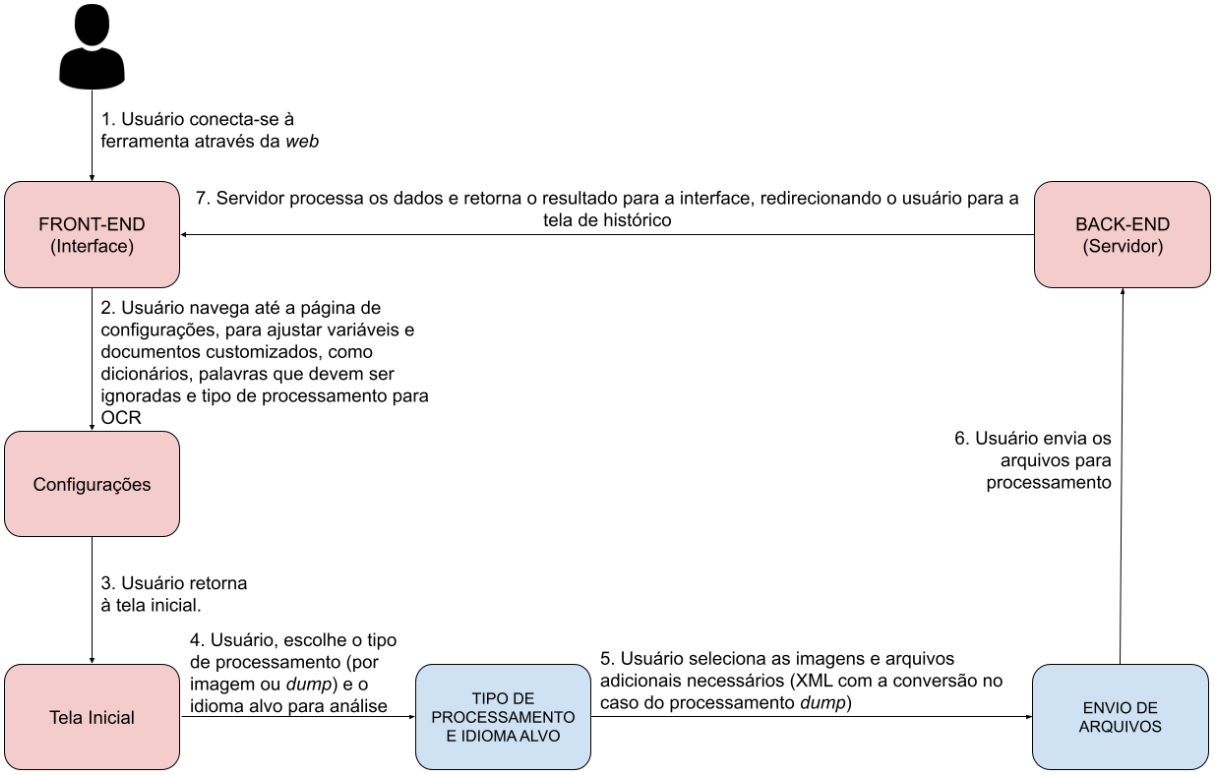

Figura 5: Fluxograma Completo da Ferramenta

4.1 BIBLIOTECAS E OUTRAS FERRAMENTAS

Algumas bibliotecas foram utilizadas para auxiliar na busca de palavras corretas conforme o dicionário da língua desejada e no Reconhecimento Óptico de Caracteres (OCR). O OCR é uma tecnologia ou método para reconhecer caracteres a partir de um arquivo, seja de imagem, PDF, mapa de bits, entre outros. Geralmente, o OCR é combinado com inteligência artificial para auxiliar na automatização e detecção de textos.

4.1.1 Google Cloud Vision API

A Vision API é uma solução da *Google Cloud Platform* (GCP), que visa principalmente automatizar tarefas de visão, simplificar análises e desbloquear *insights* úteis (14).

Neste trabalho, ela foi utilizada para OCR em arquivos de imagens (15). Com isso, pode-se analisar qualquer imagem com texto e desenvolver uma análise de possíveis erros de localização presentes.

Essa ferramenta é bastante robusta e precisa, porém, pode gerar custos indesejados (16). Daí vem a alternativa que será mostrada na próxima seção.

4.1.2 Tesseract e OpenCV

O Tesseract é um *open-source software*, desenvolvido para OCR (17).

Por ser um *software* gratuito, utilizamo-lo como alternativa da API do GCP Vision. Para ser utilizado dentro da ferramenta, deve-se incorporá-lo juntamente com o pytesseract (18) e o OpenCV (19), que oferecem suporte e abstrações para as funções e módulos presentes no Tesseract.

4.1.3 PyMultiDictionary

A biblioteca PyMultiDictionary é utilizada para obter as atualizações mais recentes no dicionário de diversas línguas (20). Assim, não será obrigatório o *upload* de dicionários pelo usuário, facilitando o uso da ferramenta.

Apesar de bastante abrangente e robusta, o processamento das palavras pode ser mais demorado. Por isso, o *import* de dicionários customizados ainda fica disponível na ferramenta, como será explicado mais adiante.

4.2 APLICAÇÃO WEB

Para melhor organização do projeto, a aplicação foi dividida em duas partes: *Front-end* e *Back-end*.

A aplicação web foi feita em inglês para universalizar o trabalho. Além disso, o trabalho de Lucena (5) foi incorporado ao sistema, ele está apresentado como processamento por extração de tela (*dump*), como será mostrado mais adiante.

4.2.1 Front-end

A interface (*Front-end*) da aplicação foi feita em React (21) e estilizado com ajuda de Tailwind CSS (22), para manter a responsividade do site.

Nela são encontradas algumas páginas e *features* que irão ajudar na navegação.

4.2.1.1 Página Inicial

Na página inicial, é possível fazer dois tipos de processamento: por imagens ou por extração de tela (*dump*). No processamento por imagens, pode-se selecionar as imagens e visualizá-las em um carrossel para pré-visualização e, a partir daí, será possível enviá-las para análise. No processamento por extração de tela (*dump*), deve-se selecionar uma imagem e o arquivo XML correspondente para envio.

Além disso, deve-se selecionar o idioma alvo do processamento. Há 20 idiomas disponíveis, entre eles:

- Alemão (**de**)
- Bengalês (**bn**)
- Chinês (**zh**)
- Coreano (**ko**)
- Espanhol (**es**)
- Francês (**fr**)
- Indiano (**hi**)
- Inglês (**pt**)
- Italiano (**it**)
- Japonês (**ja**)
- Javanês (**jv**)
- Malaio (**ms**)
- Marati (**mr**)
- Polonês (**pl**)
- Português (**pt**)
- Romeno (**ro**)
- Russo (**ru**)
- Tâmil (**ta**)
- Turco (**tr**)
- Ucraniano (**uk**)

As figuras 6, 7 e 8 exemplificam um fluxo possível:

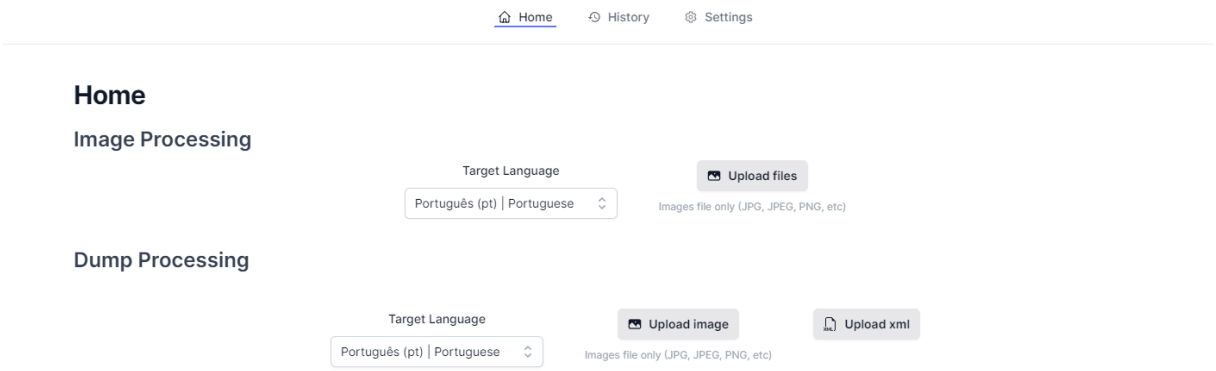

Figura 6: Página Inicial

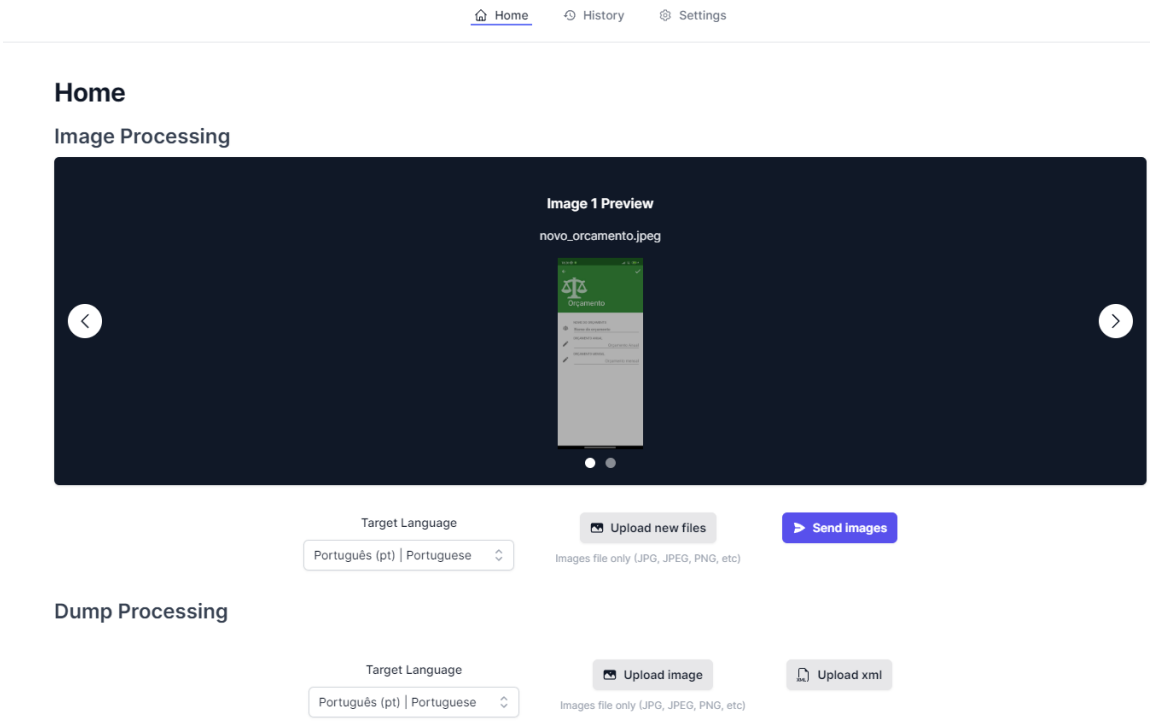

Figura 7: Página Inicial com Imagens Selecionadas. Exemplo com o Processamento de Imagens

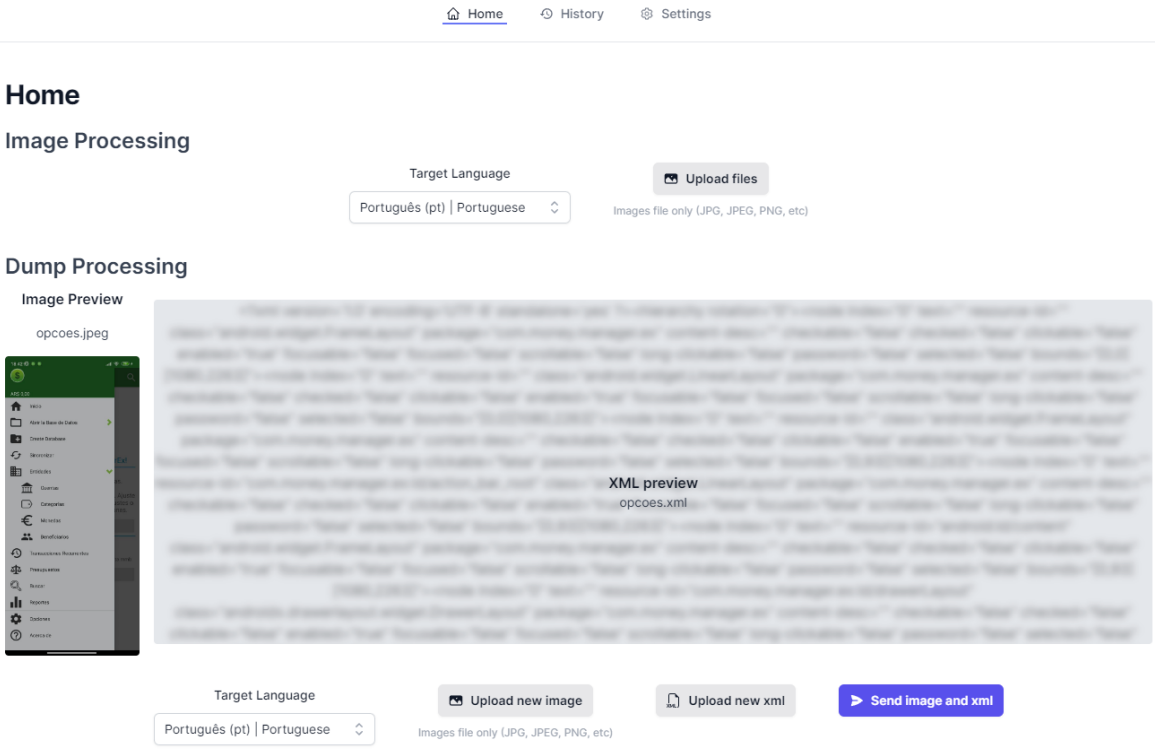

Figura 8: Página Inicial com Imagem e XML Selecionados. Exemplo com o Processamento por Extração de Tela (*dump*)

4.2.1.2 Histórico

Aqui está presente o histórico de análises realizado pelo usuário. Alguns dados são gerados pela consulta, como o ID do processamento (uuid), o idioma alvo, o tipo de processamento (processamento por imagem ou por extração de tela, também chamado de *dump*), se foi utilizado um dicionário e o tipo de uso dele (será explicado na parte de configurações), se o GCP Vision foi utilizado e o horário da requisição. Além disso, o usuário tem alguns botões disponíveis, para deletar uma análise específica, para deletar todas ou para ver o resultado do processamento selecionado.

As informações são guardadas apenas de forma local, preferencialmente por *IndexedDB* (23), mas, se indisponível, por *LocalStorage* (24). Essa forma de armazenamento é utilizada em todas as páginas e componentes da aplicação.

A figura 9 mostra um exemplo de página de histórico com um registro salvo na tabela.

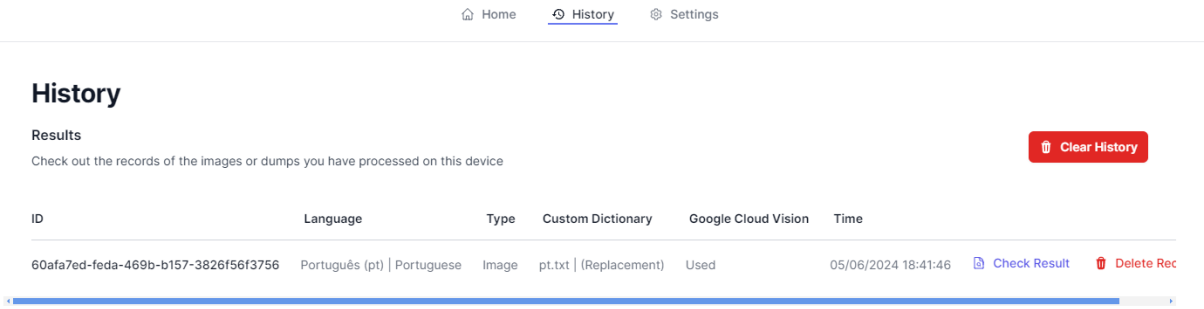

Figura 9: Página de Histórico

Ao clicar no botão "Check Result", o usuário será redirecionado a uma nova página, onde poderá ver os detalhes da análise. A figura 10 mostra um exemplo dos detalhes de uma análise.

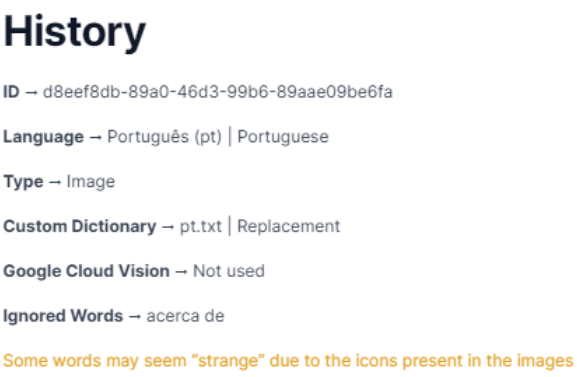

Figura 10: Página de Histórico - Detalhes da análise

As novas imagens serão marcadas com retângulos vermelhos onde forem encontrados possíveis erros. Esse processo pode gerar alguns retângulos indesejados, devido à presença de ícones.

Além disso, uma lista com as palavras incorretas encontradas fica disponível. Ao lado de cada uma há um botão para adicionar a palavra a lista de palavras que devem ser ignoradas. Se o usuário clicar neste botão, ele ficará desabilitado e uma mensagem abaixo será mostrada. As figuras 11 e 12 mostram um exemplo do resultado pós-processamento.

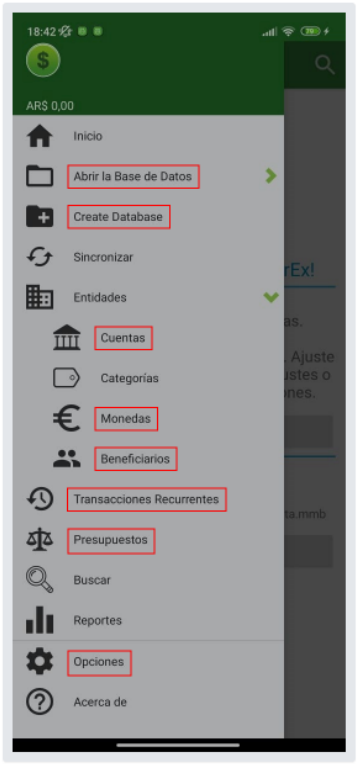

Figura 11: Página de Histórico - Imagem Pós-processamento

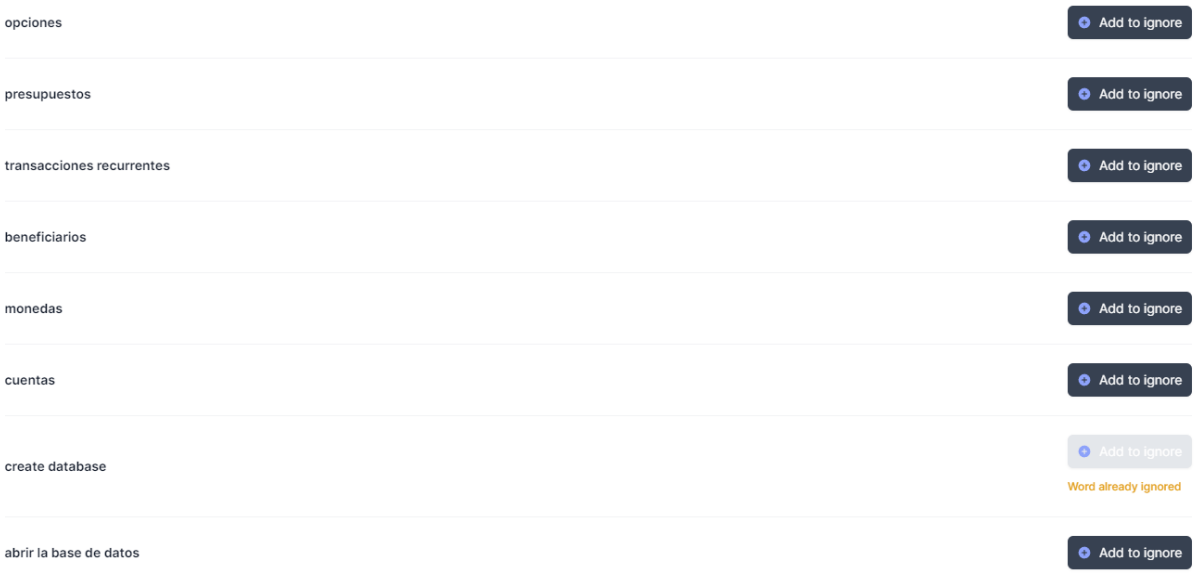

Figura 12: Página de Histórico - Palavras Incorretas

4.2.1.3 *Configurações*

Já na tela de configurações, pode-se ajustar as variáveis e arquivos de auxílio que serão utilizados nas análises. Cada uma é descrita a seguir:

- **GCP Vision**

  Aqui é possível habilitar o serviço do GCP Vision API para OCR. O usuário deve adicionar as credenciais de serviço no campo de texto presente. A figura 13 mostra um exemplo desta configuração.

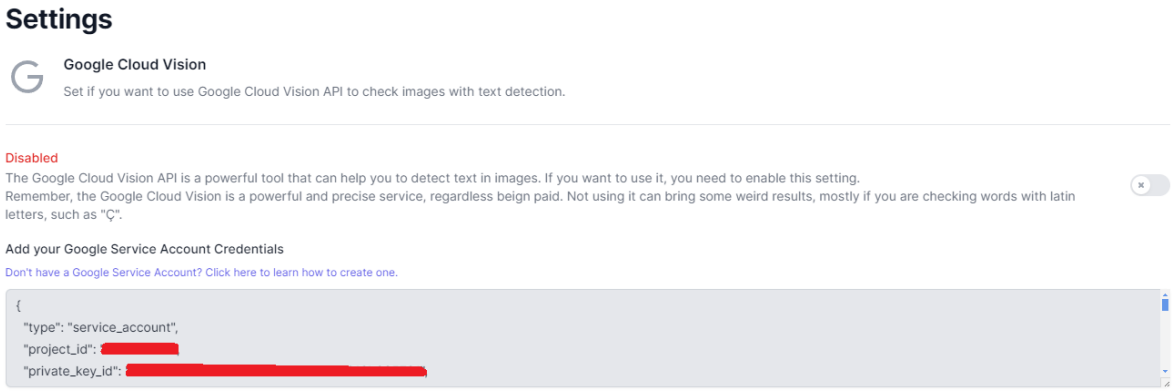

Figura 13: Página de Configurações - Google Cloud Vision API (Informações sensíveis foram censuradas)

- **Dicionários Customizados**

  O usuário pode carregar dicionários customizados e utilizá-los de duas formas:

  1. **Complemento**: Chamado de *Complement* no site, este tipo irá fazer com que o processamento utilize o dicionário selecionado **em conjunto** com o dicionário online disponível da biblioteca PyMultiDictionary.
  2. **Substituição**: Chamado de *Replacement* no site, este tipo irá fazer com que o processamento utilize **apenas** o dicionário selecionado.

  A figura 14 mostra um exemplo desta configuração.

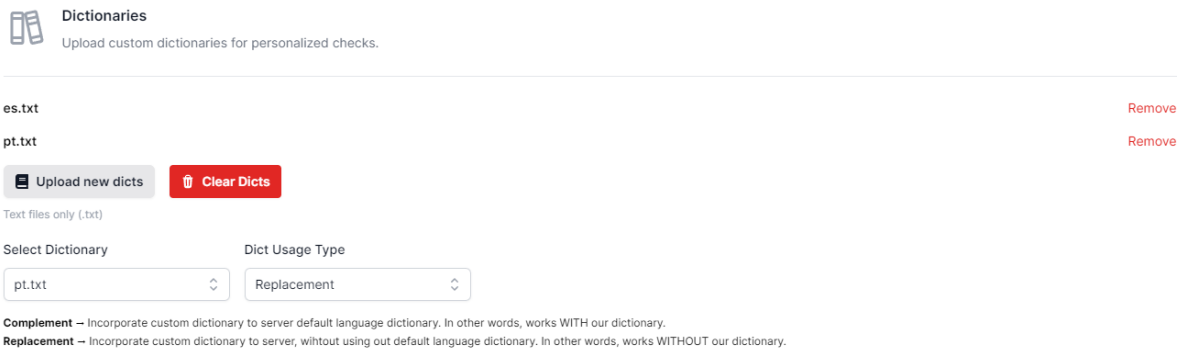

Figura 14: Página de Configurações - Dicionário

- **Palavras Ignoradas**

  O usuário pode adicionar palavras ao que é chamado de *Ignore File*, que visa montar uma lista de palavras que devem ser ignoradas nos processamentos. Estas palavras aparecem nos detalhes de cada análise no histórico. A figura 15 mostra um exemplo desta configuração.

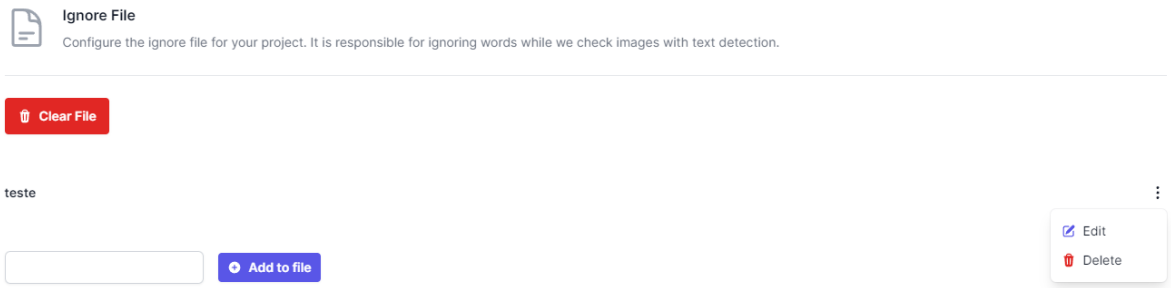

Figura 15: Página de Configurações - Palavras a serem ignoradas

- **Importação, Exportação e Exclusão de Dados**

  Por último, o usuário pode importar, exportar e/ou excluir todos os dados presentes na aplicação, incluindo o histórico. A figura 16 mostra um exemplo desta configuração.

Figura 16: Página de Configurações - Importar e exportar Configurações, e deletar todos os dados presentes

4.2.2 Back-end

O *Back-end* da aplicação foi feito em Python, utilizando a biblioteca Flask (25), onde se recebe a requisição com todos os arquivos e cada um é analisado, com base no tipo de

processamento.

Toda a ação de validar a requisição (arquivos e credenciais para o GCP, por exemplo) e desenhar os retângulos das palavras erradas ou com faltas de tradução, também é feita no *Back-end*.

4.2.2.1 Autenticação ao Google Cloud

Quando o usuário escolhe por usar o processamento via GCP Vision, ele também envia as credenciais de serviço para uso e a autenticação é realizada. O código para esta operação pode ser visualizado no Apêndice A.

4.2.2.2 Processamento dos Arquivos

O processamento por extração de tela (*dump*) é realizado de forma idêntica ao trabalho de Lucena (5), onde primeiramente as palavras são extraídas do XML correspondente com a hierarquia dos componentes e aí são verificadas. Após isso, as palavras incorretas são detectadas na imagem através do OCR.

No que diz respeito a nova alternativa, primeiramente, ao receber as capturas de tela da interface, o nosso *back-end* irá preparar a análise com base no tipo de processamento, seja via Tesseract e OpenCV ou via GCP Vision API.

Para o processamento via GCP Vision API, a imagem não precisa ser tratada e assim a análise pode ocorrer de forma mais simples. O código para este processamento pode ser visualizado no Apêndice B.

Já para o processamento via Tesseract e OpenCV, a imagem precisa ser tratada para facilitar o OCR da imagem. O código para este processamento pode ser visualizado no Apêndice C.

Para cada palavra, um algoritmo de verificação é executado, que pode ser visto no código presente no apêndice Apêndice D. Nele, retiram-se os espaços em branco no começo e final da palavra, além de ignorar dígitos, URLs e palavras presentes na lista do *Ignore File*.

Por último, desenham-se os retângulos na imagem original e é retornado a imagem codificada em base64, para facilitar a transferência de volta a interface. O código para esta operação está presente no Apêndice E.

Ao final de todas as etapas do processamento, todos os dados são retornados de volta a interface e o código HTTP, representando se a requisição obteve sucesso ou não.

5

AVALIAÇÃO

Portanto, foi possível construir uma ferramenta de detecção automática de erros de L10N. A qualidade de *software* da aplicação foi medido conforme avaliação do SUS.

Com base no estudo de Brooke (11) em 1995, uma avaliação em formato de pontuação foi realizada com base nas respostas de 9 pessoas diferentes, que possuem experiência na área de engenharia de software. Para esse cálculo, deve-se somar a contribuição de cada item e pontuar as respostas de cada usuário individualmente. Cada item terá uma contribuição que varia de 0 a 4 pontos. As perguntas ímpares (positivas) têm contribuição do seu valor menos 1, lembrando que o valor da resposta varia de 1 ("Discordo Totalmente") até 5 ("Concordo Totalmente"). Já as perguntas pares (negativas) têm contribuição de 5 menos o seu valor. Ao final, multiplica-se o valor obtido por 2,5 para obter a pontuação de cada resposta. Após isso, tira-se a média aritmética de todas as respostas e a pontuação final é obtida. Com isso, a pontuação final foi de **93,33**.

A tabela 1 mostra a resposta de cada pessoa:

Tabela 1: Pontuação por Resposta

Pessoa	P1	P2	P3	P4	P5	P6	P7	P8	P9
Pontuação	97,5	85	100	92,5	87,5	97,5	92,5	100	87,5

Segundo o trabalho de Bangor *et al.* (12) em 2009, que estabeleceu classificações de aceitabilidade conforme a pontuação alcançada, a classificação do trabalho apresentado é de Aceitável (Nota A - Excelente). A escala e classificações podem ser visualizadas na figura 17.

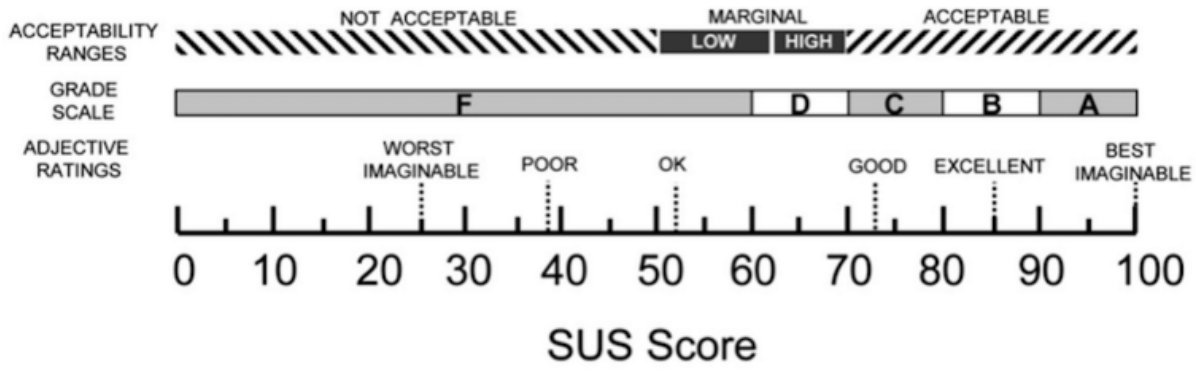

Figura 17: Classificações do SUS segundo Bangor *et al.* (2009)

6

CONCLUSÃO

Ao final, o presente trabalho mostrou uma forma de realizar análise de imagens com possíveis problemas de L10N de forma fácil, rápida e acessível, assegurando sua importância para pesquisas futuras, de modo a construir uma aplicação *web*. Além disso, o repositório do projeto está disponível no GitHub (26).

Dessa forma, a ferramenta mostrou-se satisfatória aos profissionais e desenvolvedores por conta do fácil uso e eficiência.

Para trabalhos futuros, algumas boas ideias são listadas a seguir:

- Incorporar novos tipos de análise a plataforma, como a integração com o DroidBot (27), para realizar investigações mais profundas em aplicativos Android.
- Disponibilizar a ferramenta na *web* será de extrema importância para garantir a acessibilidade a todos, uma vez que a versão de desenvolvimento pode causar problemas na execução local.
- Adicionar novas línguas ao site para globalizá-lo ainda mais (I18N).

REFERÊNCIAS

[1] GROSS, S. Internationalization and localization of software. Eastern Michigan University, Department of Computer Science (Master of Science in Computer Science), Ypsilant, Michigan, Junho, 2006. Disponível em: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=bfee27a83f63ba52a3977522bf64988b0a4b2938. Acesso em 14 jun. 2024.

[2] COUTO, M.; MIRANDA, B. An industrial experience report on the challenges in training localization and internationalization testers. In: Proceedings of the 8th Brazilian Symposium on Systematic and Automated Software Testing. P. 96–98, 2023.

[3] Google Scholar. Disponível em: https://scholar.google.com. Acesso em 04 mai. 2024.

[4] ZALIESKAITĖ, Neda. Automating detection of software cultural elements inconsistencies with locale norms. Vilnius University, Department of Mathematics and Informatics Faculty. Dissertação de Mestrado, 2023. Disponível em: https://epublications.vu.lt/object/elaba:192827410. Acesso em 20 jul. 2024.

[5] LUCENA, M. Detecção Automática de Falhas de Localização em Aplicativos Android. Universidade Federal de Pernambuco, Trabalho de Graduação, 2024.

[6] COUTO, M.; MIRANDA, B. L10n-trainer: a Tool to Assist in the Training of Localization (l10n) and Internationalization (i18n) Testers. In: Proceedings of the XXXVII Brazilian Symposium on Software Engineering. P. 277-282, 2023. DOI: 10.1145/3613372.3613420. Disponível em: https://dl.acm.org/doi/10.1145/3613372.3613420. Acesso em: 08 ago. 2024.

[7] FELIPE, L.; MIRANDA, B. Supporting Localization Testing through Automated Application Navigation. Anais Estendidos do XXII Simpósio Brasileiro de Qualidade de Software (SBQS). P. 25-30, 2023. DOI: 10.5753/sbqs_estendido.2023.235969. Disponível em: https://sol.sbc.org.br/index.php/sbqs_estendido/article/view/25911. Acesso em: 08 ago. 2024.

[8] FELIPE, L.; MIRANDA, B.; COUTO, M. TString: a tool to locate the target string's screen based on automatic exploration. Proceedings of the 9th Brazilian Symposium on Systematic and Automated Software Testing. 2024.

[9] ALAMEER, Abdulmajeed; MAHAJAN, Sonal; HALFOND, William G. J. Detecting and Localizing Internationalization Presentation Failures in Web Applications. In: 2016 IEEE International Conference on Software Testing, Verification and Validation (ICST).

P.202-212. 2016. DOI: 10.1109/ICST.2016.36. Disponível em: https://ieeexplore.ieee.org/document/7515472. Acesso em: 08 ago. 2024.

[10] SILVA, Simone Vasconcelos. Critérios da usabilidade: Um auxílio à qualidade do software. Revista Vértices, [S. l.], v. 5, n. 2, p. 111–122, 2010. DOI: 10.5935/1809-2667.20030014. Disponível em: https://editoraessentia.iff.edu.br/index.php/vertices/article/view/1809-2667.20030014. Acesso em: 20 jul. 2024.

[11] BROOKE, John et al. SUS-A quick and dirty usability scale. Usability evaluation in industry, Vol. 189, N. 194, P. 4-7, 1995. Disponível em: https://www.researchgate.net/publication/228593520_SUS_A_quick_and_dirty_usability_scale. Acesso em 14 jun. 2024.

[12] BANGOR, Aaron; KORTUM, Philip; MILLER, James. Determining what individual SUS scores mean: adding an adjective rating scale. J. Usability Studies, Vol. 4, N. 3, P. 114–123. Maio, 2009. Disponível em: https://dl.acm.org/doi/10.5555/2835587.2835589. Acesso em 14 jun. 2024.

[13] LIKERT, R. A technique for the measurement of attitudes. Archives of Psychology, 22 140, 55. 1932.

[14] Google Cloud Vision API. Disponível em: https://cloud.google.com/vision. Acesso em 04 mai. 2024.

[15] OCR com Vision API. Disponível em: https://cloud.google.com/vision/docs/ocr. Acesso em 04 mai. 2024.

[16] Preços da Vision API. Disponível em: https://cloud.google.com/vision/pricing. Acesso em 29/06/2024.

[17] Tesseract. Disponível em: https://github.com/tesseract-ocr/tesseract. Acesso em 18 mai. 2024.

[18] Python Tesseract. Disponível em: https://pypi.org/project/pytesseract. Acesso em 18 mai. 2024.

[19] OpenCV. Disponível em: https://pypi.org/project/opencv-python. Acesso em 18 mai. 2024.

[20] PyMultiDictionary. Disponível em: https://pypi.org/project/PyMultiDictionary. Acesso em 13 mai. 2024.

[21] React. Disponível em: https://react.dev. Acesso em 04 mai. 2024.

[22] Tailwind CSS. Disponível em: https://tailwindcss.com. Acesso em 04 mai. 2024.

[23] IndexedDB. Disponível em: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API. Acesso em 04 mai. 2024.

[24] LocalStorage. Disponível em: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage. Acesso em 04 mai. 2024.

[25] Flask. Disponível em: https://flask.palletsprojects.com/en/3.0.x. Acesso em 04 mai. 2024.

[26] Repositório da Ferramenta Construída. Disponível em: https://github.com/guimorone/tcc. Acesso em 06 jul. 2024.

[27] DroidBot. Disponível em: https://github.com/honeynet/droidbot. Acesso em 24/06/2024.

Apêndice

A

CÓDIGO DE AUTENTICAÇÃO DO GCP

```
from google.cloud import vision
from google.oauth2.service_account import Credentials
from utils.exceptions import BackendError

def get_google_vision_client(
        self,
    ) -> vision.ImageAnnotatorClient:
        try:
            credentials = Credentials.from_service_account_info(
                self.google_service_account_credentials
            )
            return vision.ImageAnnotatorClient(
                credentials=credentials
            )
        except Exception as err:
            raise BackendError(
                'Invalid Credentials: \n{}'.format(
                    str(err)
                )
            )
```

B

PROCESSAMENTO DAS IMAGENS VIA GCP

```
from PIL import Image
from google.cloud import vision
from utils.misc import (
    check_if_word_is_correct,
    draw_missing_words,
)
from utils.exceptions import BackendError

def google_process(
        self, screenshot: FileStorage
    ) -> Tuple[str, List[str], bytes | None]:
        client = self.get_google_vision_client()
        content = screenshot.read()
        image = vision.Image(content=content)
        response = client.text_detection(image=image)
        if response.error.message:
            raise BackendError(response.error.message)

        texts = response.text_annotations
        incorrect_words = []
        bounds = []
        for text in texts:
            word = text.description.strip().lower()
            word_splitted = word.split()
            if len(word_splitted) > 1:
                all_correct = True
                for w in word_splitted:
                    if check_if_word_is_correct(
                        w,
```

```
                    self.language,
                    self.custom_dict,
                    self.dict_usage_type,
                    self.ignore_words,
                ):
                    continue

                all_correct = False
                if w not in incorrect_words:
                    incorrect_words.append(w)
            if all_correct:
                continue
        else:
            if check_if_word_is_correct(
                word,
                self.language,
                self.custom_dict,
                self.dict_usage_type,
                self.ignore_words,
            ):
                continue

            if word not in incorrect_words:
                incorrect_words.append(word)

        bounds.append(
            [
                (vertex.x, vertex.y)
                for vertex in text.bounding_poly.vertices
            ]
        )

    drawn_image = draw_missing_words(
        Image.open(screenshot), bounds
    )

    return (
        screenshot.filename,
        incorrect_words,
```

```
            drawn_image,
        )
```

C

PROCESSAMENTO DAS IMAGENS VIA OPENCV + TESSERACT

```
import os
import cv2
import pytesseract
import numpy as np
from utils.misc import (
    check_if_word_is_correct,
    draw_missing_words,
)
from utils.exceptions import BackendError

def opencv_process(
        self, screenshot: FileStorage
    ) -> Tuple[str, List[str], bytes | None]:
        tesseract_path = os.environ.get('TESSERACT_PATH')
        if not tesseract_path:
            raise BackendError('Tesseract path is required')

        pytesseract.pytesseract.tesseract_cmd = (
            tesseract_path
        )
        content = screenshot.read()

        # CV2
        nparr = np.fromstring(content, np.uint8)
        img_np = cv2.imdecode(nparr, cv2.IMREAD_COLOR)
        gray = cv2.cvtColor(img_np, cv2.COLOR_BGR2GRAY)
        _, thresh1 = cv2.threshold(
```

```
            gray,
            0,
            255,
            cv2.THRESH_OTSU | cv2.THRESH_BINARY_INV,
        )
        rect_kernel = cv2.getStructuringElement(
            cv2.MORPH_RECT, (18, 18)
        )
        # Applying dilation on the threshold image
        dilation = cv2.dilate(
            thresh1, rect_kernel, iterations=1
        )
        # Finding contours
        contours, _ = cv2.findContours(
            dilation,
            cv2.RETR_EXTERNAL,
            cv2.CHAIN_APPROX_NONE,
        )

        incorrect_words = []
        bounds = []
        for cnt in contours:
            x, y, width, height = cv2.boundingRect(cnt)
            cropped = img_np[y : y + height, x : x + width]
            text = pytesseract.image_to_string(cropped)
            word = text.strip().lower()
            texts = word.split()
            if len(texts) > 1:
                all_correct = True
                for t in texts:
                    if check_if_word_is_correct(
                        t,
                        self.language,
                        self.custom_dict,
                        self.dict_usage_type,
                        self.ignore_words,
                    ):
                        continue
```

```
                all_correct = False
                if t not in incorrect_words:
                    incorrect_words.append(t)
            if all_correct:
                continue
        else:
            if check_if_word_is_correct(
                word,
                self.language,
                self.custom_dict,
                self.dict_usage_type,
                self.ignore_words,
            ):
                continue

            if word not in incorrect_words:
                incorrect_words.append(word)

        bounds.append(
            [
                (x, y),
                (x + width, y),
                (x + width, y + height),
                (x, y + height),
            ]
        )

    drawn_image = draw_missing_words(
        Image.open(screenshot), bounds
    )

    return (
        screenshot.filename,
        incorrect_words,
        drawn_image,
    )
```

D

ALGORITMO PARA ANALISAR UMA PALAVRA

```
import re
from typing import List, Optional, Literal
from PyMultiDictionary import MultiDictionary

def check_if_word_is_correct(
    word: str,
    language: str,
    custom_dict: Optional[List[str]],
    dict_usage_type: Literal['complement', 'replacement'],
    ignore_words: List[str] = [],
) -> bool:
    word = word.strip().lower()
    if not word or word in ignore_words or ignore_word(word):
        return True

    # double check if we ignore the word after regex sub
    word = re.sub(r'[^\w\s$]|[\d°]', '', word)
    if not word or word in ignore_words or ignore_word(word):
        return True

    dictionary = MultiDictionary()
    dictionary.set_words_lang(language)
    # Classification = [Noun, Verb, Adjective, etc].
    if custom_dict and len(custom_dict) > 0:
        if word in custom_dict:
            return True

        if dict_usage_type == 'complement':
            classification, _, _ = dictionary.meaning(
```

```
                language,
                word,
            )
            if classification and len(classification) > 0:
                return True
    else:
        classification, _, _ = dictionary.meaning(
            language,
            word,
        )
        if classification and len(classification) > 0:
            return True

    return False
```

E

CÓDIGO PARA DESENHAR PARTES COM FALHAS

```
from io import BytesIO
from base64 import b64encode
from typing import List, Optional, Tuple
from PIL import Image, ImageDraw

def draw_missing_words(
    image: Image,
    bounds: List[List[Tuple[float, float]]],
    image_format: str = 'PNG',
    fill: Optional[str] = None,
    outline: Optional[str] = 'red',
    outline_width: int = 2,
) -> Optional[bytes]:
    offset = 3
    try:
        draw = ImageDraw.Draw(image)
        for vertices in bounds:
            width, height = image.size
            if len(vertices) != 4:
                continue
            # top-left corner
            x1, y1 = vertices[0]
            if x1 > offset:
                x1 -= offset
            if y1 > offset:
                y1 -= offset
            # bottom-right corner
            x2, y2 = vertices[2]
            if x2 < width - offset:
```

```
                x2 += offset
            if y2 < height - offset:
                y2 += offset
            draw.rectangle(
                [x1, y1, x2, y2],
                fill=fill,
                outline=outline,
                width=outline_width
            )

        buffer = BytesIO()
        image.save(buffer, format=image_format)

        return b64encode(buffer.getvalue())
    except:
        return None
```