VLibras Plugin / Widget

DEV Guide

fev/2021

I. Introdução

Esse documento é um guia de desenvolvimento para entender melhor as ferramentas VLibras Plugin e VLibras Widget. Por partilharem (quase) o mesmo código, ambas as ferramentas serão discutidas juntas neste documento. Nas próximas seções, será apresentada uma introdução geral da ferramenta, arquitetura, diferenças entre si e manual de testes.

II. Breve descrição

O VLibras Plugin e Widget são ferramentas desenvolvidas em Javascript para uso de tradução automática de português para LIBRAS em páginas web, tornando-as acessíveis. Ambas possuem suporte para os navegadores Chrome, Firefox e Safari. Como personagens principais do VLibras, temos o Ícaro e a Hozana, os avatares oficiais que aparecem na ferramenta.

O VLibras Plugin, como o nome já diz, é um plugin para navegadores web, onde fica a cargo do usuário instalar e usar em qual página deseja, podendo ser instalado via webstore de cada navegador, como aqui. Já o VLibras Widget fica embarcado na própria página web, à mercê de qualquer pessoa que acesse uma página que possui a ferramenta. Atualmente, todos os sites do governo brasileiro possuem o Widget.

A estrutura do Plugin e Widget são praticamente as mesmas, com o Widget possuindo algumas peculiaridades. Para abrir o Widget, por exemplo, é necessário clicar em um ícone presente na página, e isso habilita o carregamento de toda sua estrutura no site. Já no Plugin, ele é ativado a partir de um evento de clique com o botão direito do mouse ao selecionar qualquer palavra/texto e clicar na opção do VLibras. Isso faz abrir sua janela.

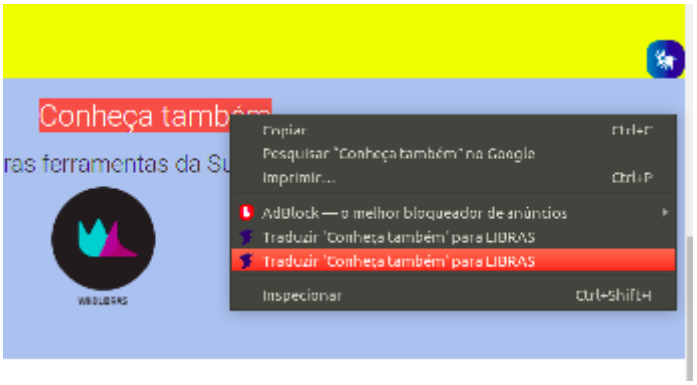

Figura 1 - Ícone no canto superior direito, relacionado Widget, e menu do navegador com função de traduzir o texto via Plugin.

Outra diferença é que o Widget é mais configurável, visto que, em seu momento de instanciação, o usuário que embarca o Widget em seu site consegue setar parâmetros referentes a personalização e opacidade da tela. A personalização é um arquivo Json, que contém propriedades referente ao avatar, como cor do cabelo, roupa, pele, como também a adição de uma logo na camisa do avatar, e a posição da mesma

(centro ou lado direito). Esse Json é enviado ao player do Widget, que fará a alteração estética do avatar em tempo real.

Em relação a opacidade, ela pode ser configurada no menu, mas também pode-se ser setada no momento de instanciação do Widget, passando o parâmetro (entre 0 e 1) relacionado ao nível de opacidade.

A configuração do ambiente, instalação das ferramentas e configuração de parâmetros, descrito brevemente nos parágrafos anteriores, podem ser encontradas no ReadMe do projeto. Já a descrição de todas as funcionalidades do Widget, pode ser encontrada em seu Manual de Instalação. Atualmente, as ferramentas estão na versão 5.2.0. Para continuar a leitura deste documento, é recomendado ler estes manuais antes.

III. Arquitetura

Por mais que as ferramentas sejam construídas em javascript, toda a lógica relacionada ao avatar, e o carregamento do avatar em si, é feito no Unity. Iniciar uma animação, pausar, aumentar a velocidade, mudança na personalização, carregamento de animações, e dentre outras funções, são responsabilidades do Unity. Na Figura 2 pode-se observar o fluxo de seleção do texto até a animação pelo avatar:

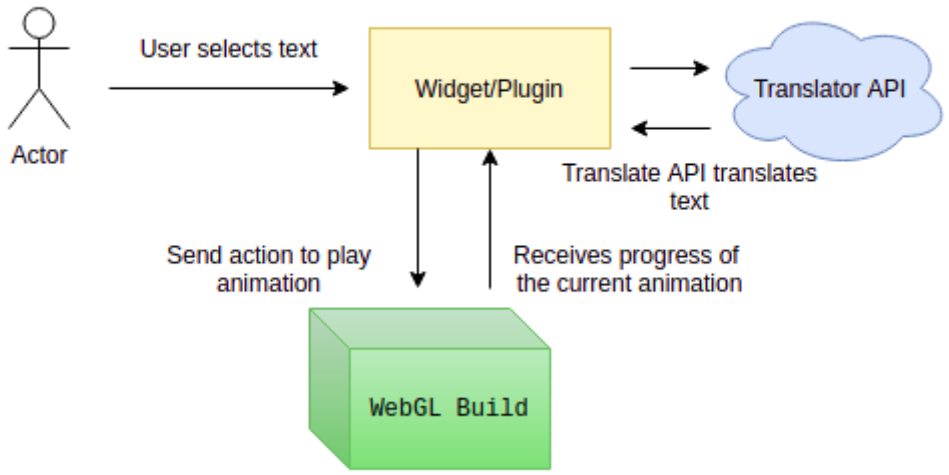

Figura 2 - Fluxo de reprodução de sinais do avatar.

1. O usuário seleciona o texto na página web.

2. O Widget/Plugin reconhece o texto selecionado e faz uma requisição POST à API de tradução para obter o texto em português em glosa.

2.1. Se ocorrer algum erro com a requisição, usa-se o texto em português.

3. Com isso, é enviado um **send** para o Player do Unity, mandando a glosa como parâmetro, com uma ação de iniciar a animação.

4. O player recebe essa informação e busca cada palavra no repositório de sinais, em busca de achar os arquivos de animações (BUNDLES) para o avatar conseguir reproduzir o sinal em LIBRAS.

4.1. Se não encontrar, o avatar irá apenas soletrar a palavra.

5. Durante a animação, o Player envia ao Widget um contador com quantos sinais já foram reproduzidos durante a animação, para o funcionamento da barra de progresso.

Para a integração do código javascript com a aplicação no Unity (Player), o Unity faz a compilação do código de forma compatível com plataformas web. Esse compilado utilizado pode ser chamado de build WebGL, e podemos vê-lo na Figura 3.

No front-end, possuímos dois repositórios relacionados ao Widget/Plugin. O primeiro é o vlibras-player-webjs, que funciona como uma lib para o vlibras-web-browsers. Nele, adicionamos os arquivos da build do Unity (src/target), e é onde ocorre toda a comunicação com o Player, funcionando como um middleware de comunicação. O envio do link do arquivo de personalização é feito via front-end, e assim essa string é enviada para o Player, sendo nele que ocorre a requisição para o servidor que contém os arquivos de animação, a fim de obter a imagem e personalizar o avatar.

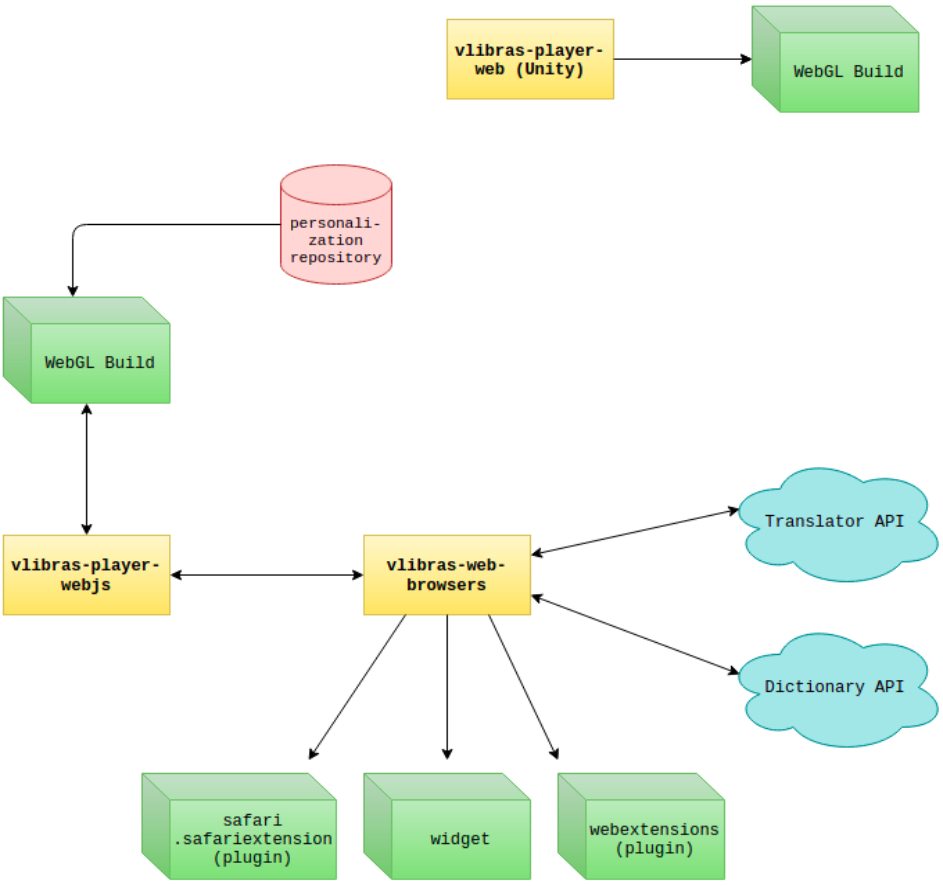

Figura 3 - Arquitetura do VLibras Plugin / Widget.

Já no vlibras-web-browsers é onde ocorre toda a criação da interface e lógica de funcionamento do Widget/Plugin ao todo. Duas pastas principais podem ser identificadas no projeto: Plugin e Widget. Na pasta Plugin, é onde ficam (quase) todos os componentes utilizados em ambas as ferramentas, como o componente de Menu, componente da tela de informações, de controles (play, pause, alterar velocidade), etc.

Na pasta Widget, ficam arquivos relacionados **exclusivamente** ao Widget, como o componente de ícone de acessibilidade que se encontra nos sites, ou a configuração de posição da tela que o Widget que é setada no menu.

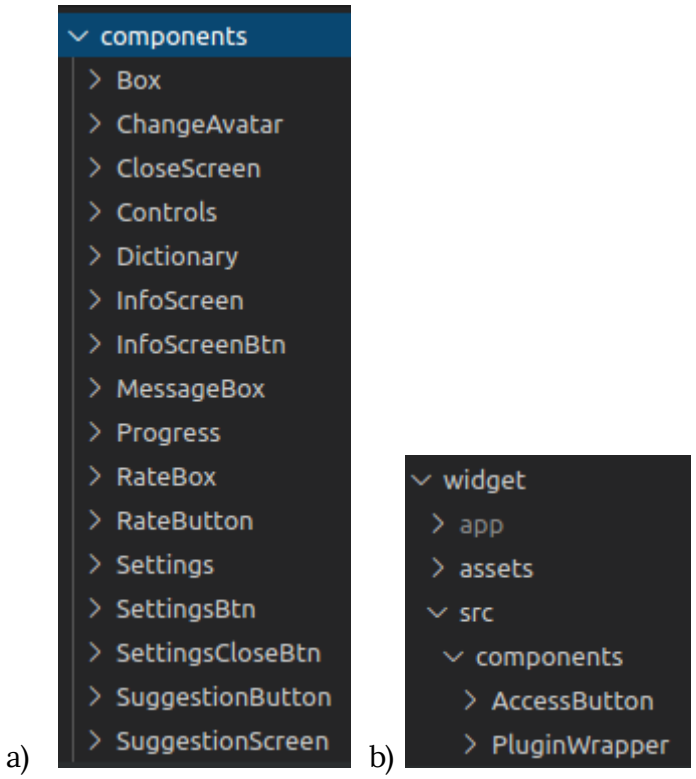

Figura 4 - a) Componentes do Widget/Plugin. b) Componentes exclusivos do Widget

Com o código pronto ou alterado, é necessário fazer uma build do código para poder testá-los. Essa build é gerada de três maneiras diferentes: para o Widget, gera a pasta **app**, e para os Plugins, para Chrome e Firefox, é gerado o **webextensions** e para o Safari o **safari.safariextension**. É gerado um arquivo compilado através de um processo de minificação de todo o código, denominado vlibras-plugin.js. As imagens e a build do Unity, durante o processo, são enviadas para pastas novas com a estrutura pré-definida, cujo arquivo minificado possui acesso. O manual de testes será apresentado na próxima seção.

Atualmente, as ferramentas se comunicam com duas APIs: a de tradução e a do dicionário. A API de tradução é utilizada para, além de fazer a tradução de texto em portugues para glosa, receber um feedback se a tradução realizada foi boa ou não. Caso a tradução não seja aprovada pelo usuário, pede-se qual a melhoria ou a tradução

correta, e assim, a informação é enviada como request. Essa sugestão de melhoria passa por um processo de sugestão, onde, para as palavras digitadas, aparecem os sinais que o repositório do VLibras Dicionário contém.

No menu contém uma tela de Dicionário, que são carregados todos os sinais existentes no VLibras Dicionário, através de requisição para sua API, podendo-os ser consultados um a um.

IV. Manual de Testes

Primeiramente, é necessário ter o node instalado em sua máquina. Após isso, é necessário clonar o repositório vlibras-player-webjs e fazer a instalação de seus módulos. No repositório já possui arquivos de build do Unity, então não é necessário rodar a ferramenta Unity para utilizar o Plugin/Widget. Sendo assim, é necessário gerar apenas a pasta de build no repositório atual, para ser utilizado como lib dentro dos node_modules do repositório vlibras-web-browsers.

É preciso rodar o segundo comando a cada vez que atualizar os arquivos de build do Unity, ou seja, houver alteração no Player.

```
$ npm install
$ npm run build
```

Após isso, na mesma pasta onde foi clonado o vlibras-player-webjs, deve-se clonar o vlibras-web-browsers e instalar as dependências. Com isso, para gerar os arquivos de build de Plugin e Widget, passamos parâmetros distintos no terminal.

```
$ npm install
$ npm run gulp build:widget
$ npm run gulp build:safari
$ npm run gulp build:webextensions
```

E caso queira fazer a build completa, tem-se a opção:

```
$ npm run gulp build
```

1) Widget

Com as builds concluídas, podemos testá-las. Para testar o Widget, deve-se rodar o comando abaixo.

```
$ npm run gulp run:widget
```

Com isso, você deve abrir em seu navegador o endereço http://localhost:8080/, carregando, assim, a página de teste, com o ícone clicável no canto direito. Caso você queira testar em sua própria página web, tem-se duas opções:

a) Mover a pasta app para o diretório de sua página web, e, na instanciação do Widget, passar como parâmetro o diretório da sua pasta app personalizada (vê-se no Readme e no Manual de uso como passar).
b) Em vez de passar o diretório como parâmetro e usar uma pasta local, utilizar a última versão estável do Widget, hospedada em https://vlibras.gov.br/app/ . Apenas enviar a url como parâmetro.

Dentro da pasta app contém a página de teste intitulada index.html, e você pode utilizá-la como modelo.

2) Plugin

O processo de teste dos plugins é mais trabalhoso, visto que é diretamente no navegador. Nesse exemplo, será apresentado o teste da build webextensions, usando o navegador Google Chrome.

O primeiro passo é ir na tela de extensões, e ativar o modo desenvolvedor no canto superior direito. Com isso, deve-se ir em **Carregar sem compactação** e selecionar a pasta **webextensions**, visto que é a build compatível com o Google Chrome. Após isso, seu plugin estará atualizado, e é só selecionar um texto, clicar com o botão direito e ir em Traduzir para LIBRAS.

Obs.: Ao modificar o código, se ele estiver rodando, para toda alteração em arquivos do projeto será gerado uma build nova do Widget automaticamente, e assim, não é necessário rodar o comando toda vez que estiver desenvolvendo e for testar.

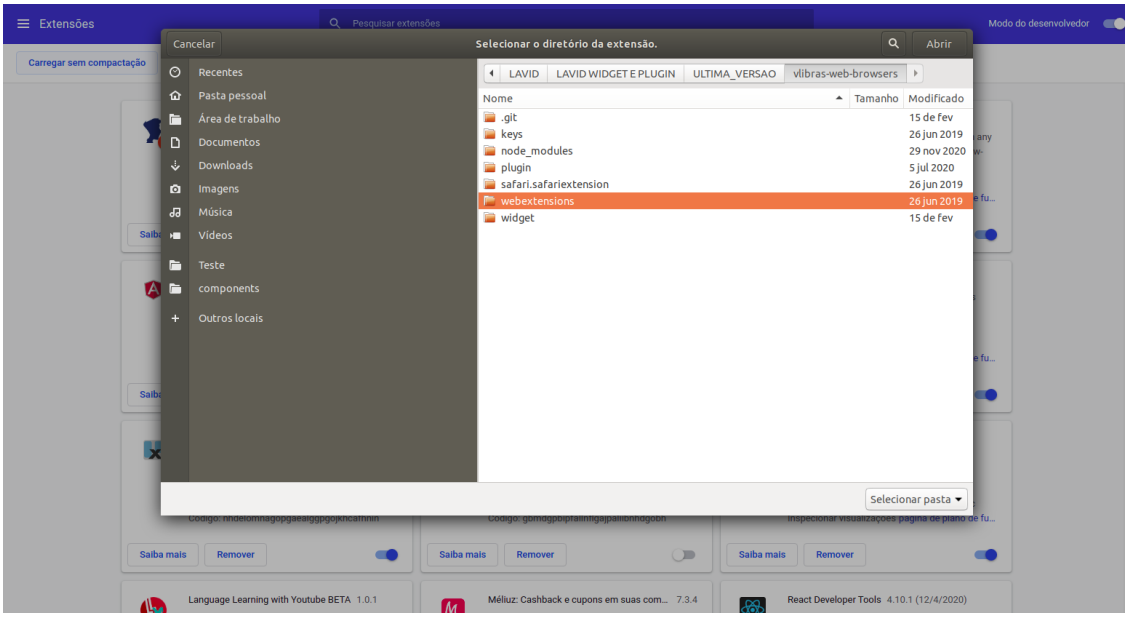

Figura 5 - a) Carregamento de build do Plugin no Google Chrome.