CRIAÇÃO DE UM *FRAMEWORK FRONT-END* PARA O DESENVOLVIMENTO DE APLICAÇÕES *WEB* RESPONSIVAS COM ACESSIBILIDADE PARA DEFICIENTES VISUAIS

Renan Soares Germano (IC) e Maria Amelia Eliseo (Orientador)

Apoio:PIBITI Mackenzie

RESUMO

O presente artigo apresenta o estudo da acessibilidade e responsividade na elaboração de páginas *web* para a criação de um *framework* que irá auxiliar no desenvolvimento destas páginas. O *framework* prevê a criação de aplicações *web* responsivas, com acessibilidade para deficiente visuais, utilizando as tecnologias HTML5, CSS3 e JavaScript. É apresentada a ferramenta desenvolvida, seu funcionamento, alguns exemplos de uso e suas contribuições. Como resultado final, o *framework* desenvolvido proporciona melhorias em relação a responsividade e acessibilidade, mas que, no entanto, ainda devem ser formalmente validados em possíveis trabalhos futuros.

Palavras-chave: Acessibilidade. Responsividade. Aplicações Web.

ABSTRACT

This paper presents the study of accessibility and responsiveness in the elaboration of web pages for the creation of a framework that will assist in the development of these pages. The framework foresees the creation of responsive web applications with accessibility for the visually impaired, using the technologies HTML5, CSS3 and JavaScript. The tool created, its operation, some usage examples and their contributions are presented. As a final result, the developed framework provides improvements regarding responsiveness and accessibility, but which, however, have yet to be formally validated in possible future works.

Keywords: *Accessibility. Responsiveness. Web applications.*

1. INTRODUÇÃO

De acordo com dados da Organização Mundial da Saúde (OMS), estima-se que 15% da população possui algum tipo de deficiência, e esse número está crescendo, entre outros motivos, devido ao avanço da idade da população e ao aumento de condições crônicas de saúde (WORLD HEALTH ORGANIZATION, 2018a).

Também de acordo com a OMS, é estimado que 1,3 bilhão de pessoas vivem com algum tipo de deficiência visual. Dentre elas, 188,5 milhões possuem baixa visão, 217 milhões possuem deficiência visual moderada ou severa e 36 milhões são cegas (WORLD HEALTH ORGANIZATION, 2018b).

No Brasil, de acordo com dados do censo de 2010, realizado pelo Instituto Brasileiro de Geografia e Estatística (IBGE), 6,7% da população declarou ter algum tipo de deficiência nas habilidades investigadas (enxergar, ouvir, caminhar ou subir degraus), o que correspondia a 12,5 milhões de brasileiros. Também foi constatado que dentre as pessoas que declararam ter algum tipo de deficiência, 3,4% declararam ter alguma deficiência visual (IBGE, 2010).

A deficiência é uma questão de direitos humanos, pois as pessoas com deficiência devem ter assegurados os mesmos direitos e oportunidades que uma pessoa que não possui deficiência. Por isso, cabe às sociedades e aos governos garantirem que essa população seja devidamente tratada e respeitada. O tema tem sido discutido em âmbito global, levando à criação de legislações que já foram adotadas por vários países. Com isso, surge o conceito de acessibilidade, que tem por objetivo adaptar os lugares, serviços, meios de comunicação, educação, etc, para que as pessoas com deficiência também tenham acesso a eles (BRASIL, 2012).

Também é importante dizer que ao longo do tempo o próprio conceito de deficiência vem sendo reinterpretado. Hoje, não se diz mais que as pessoas são deficientes, mas sim os lugares, serviços, meios de comunicação, entre outros, que não disponibilizam uma forma de acesso ou utilização para as pessoas com deficiência (BRASIL, 2012).

É inegável a grande presença da tecnologia na vida de todos hoje. Segundo pesquisa publicada em outubro de 2018 pelas empresas de *marketing* digital *We Are Social*, do Reino Unido, e Hootsuite, dos Estados Unidos, cerca de 4,176 bilhões de pessoas (55% da população mundial) possuem acesso à *internet*. A pesquisa também mostrou que 3,397 bilhões de pessoas (44% da população mundial) são usuárias ativas das mídias sociais, e 5,135 bilhões de pessoas (68% da população mundial) utilizam dispositivos móveis, entre

outros dados que deixam claro a grande presença da tecnologia nos dias de hoje (WE ARE SOCIAL, 2018).

Outro estudo que também expressa os grandes números da tecnologia na contemporaneidade, dessa vez no Brasil, é a Pesquisa Anual do Uso de TI - realizada pela Fundação Getúlio Vargas (FGV) -, que em sua 29ª edição realizada em 2018 mostra que existem 174 milhões de computadores em uso no país - 5 computadores (considerando-se *desktops*, *notebooks* e *tablets*) a cada 6 habitantes. O levantamento ainda prevê que entre os anos de 2020 e 2022 a proporção de computadores por habitante será de um para um, e que no ano da pesquisa existiam 220 milhões de *smartphones* no país (mais de um por habitante) (FUNDAÇÃO GETÚLIO VARGAS, 2018).

Diante desse cenário, fica claro que é importante prover meios de acesso facilitadores para que as pessoas com deficiência possam e consigam desfrutar dos benefícios proporcionados pela tecnologia. Para tanto, existem as Tecnologias Assistivas (TAs) - *softwares* e dispositivos que proveem acessibilidade para pessoas deficientes utilizarem as tecnologias diversas de modo a não serem prejudicadas, tendo uma experiência igual ou muito semelhante a de um usuário comum (ASSISTIVA, 2019).

Quando fala-se da utilização da *internet*, a acessibilidade se configura principalmente em possibilitar meios para que as pessoas com deficiência consigam navegar nos *web sites*. Assim, é importante dar suporte para navegação com teclado e *mouse*, pensar na disposição e desenvolvimento da estrutura da página para que os usuários de leitores de tela também consigam navegar no *site*, levar em consideração o tamanho dos conteúdos, o contraste, entre outros fatores (HENRY, ABOU-ZAHRA, BREWER, 2014).

Outro ponto que também pode ser levantado em relação aos dados apresentados refere-se à diferente gama de dispositivos utilizados pela população (*desktops*, *notebooks*, *tablets* e *smartphones*), e está relacionado à usabilidade e experiência dos usuários no geral, não somente para os que necessitam das TAs: é a responsividade. Visto que todos esses aparelhos possuem acesso à *internet* e possuem diferentes tamanhos de tela, pode ser que um mesmo *site* que seja usado em um dispositivo com tela maior não proporcione suporte para utilização em telas menores. Daí entra a responsividade, que consiste em ajustar o *layout* das páginas *web* para diferentes tamanhos de tela, proporcionando a melhor experiência possível para os usuários (CARVER, 2014).

A fim de padronizar a acessibilidade na *web*, a W3C (World Wide Web Consortium) - organização internacional que cria padrões e guia o desenvolvimento da *web* para que atinja seu potencial máximo - criou diretivas de acessibilidade, que devem ser seguidas para que a

web se torne realmente acessível, como é o caso dos padrões WCAG 1.0 (W3C, 1999) e WCAG 2.0 (W3C, 2008). A WAI (*Web Accessibility Initiative*) (W3C, 2018a) também é uma iniciativa da W3C que cria padrões e *frameworks* para o desenvolvimento de *websites* acessíveis. Em âmbito nacional há o e-MAG, cartilha com diretrizes de acessibilidade do governo eletrônico (BRASIL, 2014).

Além das diretivas de acessibilidade, em muitos países a acessibilidade na *web* também foi constitucionalizada na forma de lei. Como exemplo, tem-se a seção 508, lei americana de 1998 (uma das primeiras a serem criadas, e que serviu de base para muitas outras leis e diretivas posteriores) que tornou obrigatório a adequação das Tecnologias de Informação e Comunicação (TICs) de organizações federais para serem acessíveis (SECTION508.GOV, 2018). No Brasil, existem duas leis que definem, entre outros fatores, a obrigatoriedade da acessibilidade dos meios de comunicação e acesso a informação de empresas e organizações públicas. Essas são as leis número 10.098 (BRASIL, 2000) e número 13.146 (BRASIL, 2015). É possível ter acesso à uma lista de países que possuem leis que regulam a acessibilidade na *web* na página da W3C (W3C, 2018b).

Já para a responsividade não existe um padrão internacional, porém existem técnicas e práticas de desenvolvimento que devem ser adotadas pelos desenvolvedores para que a responsividade seja obtida. Alguns fatores relacionados a esse tipo de página *web* são: o uso das ferramentas HTML5 e do CSS3, necessários para o seu desenvolvimento; a importância na consistência em diferentes dispositivos; o aumento da agilidade no uso; a maior adaptabilidade a novas tecnologias (novos dispositivos com diferentes tamanhos de tela) (CARVER, 2014).

Adicionar a acessibilidade no desenvolvimento *web* não é uma tarefa simples, pois exige que os desenvolvedores estudem e saibam como utilizar os padrões existentes. Idem para a responsividade, que exige conhecimento de técnicas específicas. Com isso, tem-se que o tempo de desenvolvimento pode crescer muito devido à essas duas exigências. Por isso, não é raro às vezes em que o desenvolvedor opta por não levar em consideração a acessibilidade no desenvolvimento. Já para o problema da responsividade, a solução encontrada é a utilização de bibliotecas de terceiros, que possibilitam a implementação de tal funcionalidade aos *web sites* de maneira mais fácil, para o desenvolvimento do que se deseja. Outro beneficio desses *frameworks* é a formatação padrão dos componentes, que geralmente é simples e agradável, exigindo pouco ou nenhum ajuste pelos desenvolvedores.

Contudo, tais *frameworks* deixam a acessibilidade em segundo plano e também se utilizam de *tags* não semânticas para a criação do *layout* responsivo, o que pode prejudicar a experiência de usuários que utilizam leitores de tela, além de complicar a manutenção do código.

Dessa maneira, a presente pesquisa visou o estudo da acessibilidade e da responsividade em páginas *web* para a criação de um *framework front-end* que auxilia a criação de aplicações *web* que possuam as seguintes características: agilidade no desenvolvimento (possibilidade de reutilização de componentes e facilidade de manutenção, bem como menor tempo de criação), responsividade e acessibilidade para deficientes visuais (implementação de mecanismos que facilitem a navegação por teclado e através dos leitores de tela, e a utilização das *tags* semânticas do HTML5).

2. REFERENCIAL TEÓRICO

2.1 Trabalhos correlatos

Em busca de trabalhos com uma proposta igual ou semelhante à desta pesquisa, encontrou-se o trabalho desenvolvido por MAHMUD (2016), cuja ideia inicial era criar uma ferramenta para o desenvolvimento de jogos acessíveis na *web*, mas que posteriormente foi decidido ampliar seu funcionamento para o desenvolvimento de páginas *web* em geral. O *framework* apresentado funciona a partir da marcação das *tags* HTML com classes específicas que tornam possível sua manipulação via JavaScript, possibilitando adicionar propriedades do WAI-ARIA às mesmas. Esse trabalho foi fundamental para a construção de uma ideia inicial do *framework* implementado.

Outro exemplo é o trabalho de JUNIOR (2013) que apresenta uma análise de acessibilidade para deficientes visuais dos *softwares* leitores de tela - um dos tipos de TAs utilizadas pelos deficientes visuais. Esse trabalho foi importante para esta pesquisa porque apresentou os principais leitores de tela utilizados atualmente, comparando-os através de diferentes testes. Além disso também foi possível ter uma visão sobre como é o suporte de cada navegador aos leitores de tela.

O *framework* turretcss (TURRETCSS, 2019) foi outro trabalho que deu suporte à esta pesquisa. Segundo sua documentação, ele foi construído para tratar a acessibilidade de páginas *web*. Contudo, o *framework* utiliza apenas CSS, e implementações mais avançadas para acessibilidade ficam por responsabilidade dos desenvolvedores, sendo necessário adicionar manualmente marcações do WAI-ARIA ao HTML.

2.2. Diretrizes de acessibilidade

A WCAG é um documento criado pela W3C que apresenta diretrizes para padronizar a acessibilidade aos conteúdos de páginas *web*. A partir dela, é possível classificar as páginas *web* em três níveis diferentes de acessibilidade: A, AA e AAA (sendo o primeiro o mais baixo, e o último, o mais alto). Essa classificação também pode ser usada para dizer o quanto uma página *web* implementa esse padrão (W3C, 2009).

Em sua versão primária, a WCAG 1.0 possui 14 diretrizes. Cada diretriz apresenta uma lista com itens que especificam o tópico abordado. Cada item dessa lista possui um nível de prioridade: 1 (tópicos que devem ser implementados para que a página seja minimamente acessível), 2 (tópicos que deveriam ser implementados para que barreiras significativas de acesso à informação sejam eliminadas) e 3 (tópicos que talvez possam ser implementados para melhorar ainda mais a acessibilidade da página). Essas prioridades são utilizadas para definir a classificação da página: páginas que implementam todos os tópicos com prioridade 1 são classificadas como A; páginas que implementam todos os tópicos com prioridades 1 e 2 são classificadas como AA; e páginas que implementam todos os tópicos com prioridades 1, 2 e 3 são classificadas como AAA (W3C, 1999).

Já na WCAG 2.0, existem 12 diretrizes, subdivididas em 4 princípios de acessibilidade: perceptível, operável, compreensível e robusto. Cada diretriz possui critérios de sucesso de nível A, AA ou AAA. Uma página é classificada como A, AA ou AAA se todos os critérios de sucesso dos respectivos níveis forem adotados (W3C, 2008).

Além das diferenças já citadas entre a WCAG 1.0 e a WCAG 2.0, esta última apresenta as seguintes vantagens: maior variedade de tecnologias abordadas, bem como técnicas avançadas; foi pensada para suportar o desenvolvimento de novas tecnologias no futuro; seus requerimentos são melhores definidos, tornando mais fácil a realização dos testes; e a quantidade de documentos complementares é maior, tornando mais fácil o desenvolvimento das páginas web seguindo as normas. Visto que a versão 2.0 é uma evolução da versão 1.0, os *sites* que implementam as diretrizes da versão 2.0 também implementam as da outra (W3C, 2009). Também existe a versão 2.1, que trouxe novos critérios de sucesso em relação a versão anterior (W3C, 2018c).

A WCAG é importante porque é um padrão internacional e pode ser usado tanto pelos desenvolvedores de páginas web, quanto por instituições e governos para estabelecerem seus próprios requisitos de acessibilidade. Idealmente, deve-se planejar o desenvolvimento de páginas *web* já levando em consideração as diretrizes de acessibilidade. No entanto, também é possível adequar uma página que não é acessível aos

padrões de acessibilidade. Em ambos os casos, a adequação com o padrão é verificada através da checagem de cada um dos pontos presentes nas documentações, através de testes que podem ser feitos por verificação humana, ou automatizados por *softwares*. Além das documentações complementares da WCAG, que apresentam técnicas e exemplos de implementação (W3C, 2018d), outro recurso que pode ajudar os desenvolvedores é o WebAIM (2018), que apresenta uma *checklist* que pode ser usada como auxilio para os desenvolvedores.

O WAI-ARIA é uma especificação técnica, criada pela W3C, que disponibiliza um *framework* que possibilita melhorar a acessibilidade e interoperabilidade dos conteúdos das páginas de aplicações *web*. É constituído de marcações que devem ser usadas nas *tags* HTML. Existem três tipos de marcação: *role* (função) - usado para definir qual o tipo de componente que se está criando; *states* (estados) e *properties* (propriedades) - usados para definir informações complementares sobre o componente, sendo que seus valores podem mudar dinamicamente. Essas marcações permitem definir o que é chamado semântica dos conteúdos. Através delas, as tecnologias assistivas - como os leitores de tela, por exemplo - são capazes de informar para o usuário o que há na tela, como interagir com o que está lá, em qual estado se encontra e o notificar caso algo mude no componente em resposta à uma ação do usuário. Elas são importantes porque melhoram a acessibilidade dos conteúdos nas páginas *web*, tornando melhor a experiência de usuários que utilizam diferentes tipos de TAs (W3C, 2017a).

Em sua documentação, é descrito que o WAI-ARIA pode ser usado para definir toda a semântica de uma página *web*, ou pode ser utilizado em conjunto com outras tecnologias que também promovam a semântica adequada dos conteúdos para as tecnologias assistivas. Um exemplo do último caso é a linguagem HTML5, que já possui uma série de *tags* semânticas, tornando desnecessário a utilização de algumas marcações do WAI-ARIA. Contudo, as marcações do WAI-ARIA podem ser utilizadas de maneira complementar para refinar a semântica, melhorando ainda mais a acessibilidade das páginas (W3C, 2017a).

O e-MAG é uma adaptação dos padrões internacionais (WCAG 1.0 e WCAG 2.0) para as necessidades de acessibilidade das páginas *web* do governo brasileiro. Sua versão atual, 3.1, apresenta diversas melhorias em relação às versões anteriores, destacando-se o maior número de exemplos de implementação dos padrões utilizando HTML5 e o WAI-ARIA. Algumas diferenças importantes com relação aos padrões internacionais são: o desmembramento das diretrizes em várias normas divididas em 6 sessões (marcação, comportamento, conteúdo/informação, apresentação/*design*, multimídia e formulário); e o

abandono da classificação em diferentes níveis (A, AA e AAA), porque os *sites* do governo devem implementar todas as recomendações, tornando-os acessíveis para o maior público possível (BRASIL, 2014).

Para sua implementação, são definidos três passos: (1) seguir os padrões *web*, (2) seguir as diretrizes ou recomendações de acessibilidade e (3) realizar a avaliação de acessibilidade. O documento também disponibiliza uma série de documentos complementares que podem ser utilizados na implementação das normas existentes. O e-MAG é importante pois foi construído com a ajuda de muitos especialistas, além de consultas públicas com a população, sendo assim fundamental em âmbito nacional, contribuindo para a inclusão social de pessoas com deficiências ao disponibilizar normas que atendam às necessidades de acessibilidade da população brasileira (BRASIL, 2014).

2.3. Páginas *Web* Responsivas

Carver (2014) define um *site* responsivo como sendo aquele que pode ser acessado através de uma mesma URL em diferentes dispositivos (*smartphones*, *tablets*, *desktops*). Além disso, também são definidas algumas características e requisitos para sua implementação: devido ao fato dos *sites* responsivos necessitarem da utilização de *media queries* para ajustar o *design* de acordo com a largura da página, faz-se necessário a utilização do CSS3 e do HTML5; um *site* responsivo deve manter a consistência de seu conteúdo nos diferentes dispositivos; os *sites* responsivos proporcionam uma melhor experiência para os usuários; e os *sites* responsivos também se adaptam mais facilmente a novas tecnologias que venham surgir (novos dispositivos com novos tamanhos de telas).

Para o seu desenvolvimento, o modelo de criação de *software* em cascata torna-se ineficiente, porque as páginas responsivas estão ligadas ao conceito de adaptação. Com isso, o *layout* definido pode mudar durante sua criação, com objetivo de buscar a melhor experiência para o usuário em diferentes dispositivos. Isso faz com que seja necessário revisitar outras partes do processo de desenvolvimento para se implementar um novo requisito (nesse caso as fases de desenvolvimento e teste para validar a nova implementação), tornando assim um modelo iterativo e cíclico mais ideal para a criação dessas páginas (CARVER, 2014).

Com relação à implementação técnica, são definidos dois tópicos principais: os *breakpoints* e as *media queries*. Os *breakpoints* consistem em definir diferentes *layouts* para diferentes tamanhos de tela, bem como definir até que tamanho de tela cada *layout* será exibido. *Media query* é a tecnologia que permite a implementação dos diferentes *layouts* definidos de acordo com o tamanho da tela - isso é feito ao se aplicar diferentes estilos à

página de acordo com informações como largura da tela e tipo de dispositivo que está sendo utilizado.

3. METODOLOGIA

Inicialmente, visando compreender melhor o funcionamento da acessibilidade para deficientes visuais ao utilizar *web sites*, foram realizados alguns testes nos navegadores (Google Chrome, Firefox e Safari) para uma navegação utilizando o teclado. Os resultados mostraram que o navegador no qual a navegação com o teclado melhor funciona é o Google Chrome.

Na sequência, foi realizado um teste com leitor de tela para navegação em páginas *web*. Ele consistiu em realizar tarefas nas redes sociais Facebook e WhatsApp. O objetivo era enviar uma mensagem para outro usuário pela *web*. Foi utilizado o VoiceOver (leitor de tela nativo dos computadores da Apple). Esse teste não teve como objetivo fazer um levantamento qualitativo ou quantitativo dos *sites*, dos navegadores ou dos leitores de tela, mas sim proporcionar uma experiência de como se utiliza um leitor de tela e se sentir como uma pessoa que precisa de acessibilidade. Ele foi importante para compreender a necessidade do usuário com deficiência visual e identificar os requisitos de acessibilidade que serão listados em seguida.

Os requisitos definidos foram:

Acessibilidade: (1) possibilitar a navegação pela página com a utilização do teclado com a tecla *tab;* (2) utilizar as marcações do WAI-ARIA quando possível para melhorar a experiência de usuários de leitores de tela; (3) criar componentes que sigam os padrões definidos pela W3C.

Utilização de HTML5 e CSS3: (4) utilizar as *tags* semânticas do HTML5 corretamente**;** (5) não utilizar *tags* do tipo *div* aninhadas para criar os *layouts*; (6) utilizar as variáveis do CSS3 para possibilitar a manipulação do tema (conjunto de cores) do *site*; (7) utilizar o CSS3 para posicionar os componentes sem a necessidade de criar *tags* não semânticas aninhadas; (8) utilizar *medias queries* para ajustar os componentes em diferentes tamanhos de tela.

Agilidade no desenvolvimento: (9) possibilitar a criação de páginas responsivas e acessíveis de modo que o desenvolvedor não necessite ter nenhum trabalho a mais do que somente estruturar a página; (10) possibilitar reutilização dos elementos.

Além disso também foi definido uma lista de componentes personalizados que poderiam ser desenvolvidos. São eles:

***Skip links*:** componente que possibilita pular direto para uma das seções do *site* sem necessidade de passar por todos os conteúdos anteriores, útil para usuário de leitores de tela.

Menu: lista de *links* que permite a navegação no *site*.

***Card*:** componente para mostrar informações de uma maneira simples.

***Alert*:** componente para exibir diferentes tipos de mensagens para os usuários.

***Layouts* prontos:** definições de *layouts* que poderiam ser utilizados pelos desenvolvedores como ponto de partida, tornando o desenvolvimento mais rápido**.**

Temas prontos: conjuntos de cores que poderiam ser utilizados pelos desenvolvedores como ponto de partida, tornando o desenvolvimento mais rápido.

Tema ajustável: possibilidade de configurar facilmente as cores que seriam utilizadas no *site*.

4. RESULTADO E DISCUSSÃO

Baseado nas diretrizes do WCAG 2.0 (W3C, 2008), do e-MAG (BRASIL, 2014), no *framework* WAI-ARIA (W3C, 2018a) e nos conceitos de responsividade de Carver (2014) foi desenvolvido um *framework front-end* para a criação de aplicações *web* responsivas com acessibilidade para deficientes visuais. Seu funcionamento se dá a partir da importação dos arquivos JavaScript (**framework.js**) e CSS (**framework.css**) na página HTML que será criada, e da implementação de algumas funções que devem retornar uma lista de objetos que representam os conteúdos que serão traduzidos em *tags* e constituirão a página.

Sua implementação, documentação, e exemplos de utilização podem ser encontrados no repositório de arquivos remoto no endereço https://github.com/rsg73626/framework-frontend-responsivo-acessivel.

O *framework* adota uma forma de criação de páginas *web* não tradicional, na qual, para a criação da página em si, não é necessário escrever em HTML, ficando esse trabalho por conta do *framework*. Essa abordagem foi adotada pensando na facilidade para os desenvolvedores de aplicações *web*, tornando possível a criação das páginas através do uso de uma linguagem de programação (JavaScript), ao invés de uma linguagem de marcação de texto (HTML).

Sendo assim, a maior parte do trabalho do desenvolvedor que utilizar a ferramenta será o de configurar os objetos de acordo com o padrão reconhecido pelo *framework* para que tudo funcione como ele deseja.

Além disso, essa abordagem também possibilita a reutilização dos componentes criados, através do compartilhamento de um mesmo objeto para a criação de diferentes trechos da página, bem como da possibilidade de compartilhar um mesmo arquivo JavaScript para construir um mesmo componente em diferentes páginas.

A abordagem também trouxe ganhos semânticos para as páginas, pois através do mecanismo de funções foi possível controlar quais os tipos de *tag* – marcadores que possuem uma notação específica utilizados para renderizar os diferentes tipos de componentes de uma página *web* nos navegadores (W3C, 2017b) - possam ser criadas, bem como quais as *tags* poderiam ser inseridas dentro de outras *tags*. Como exemplo, pode-se citar o caso das *tags* <section> e <article>. Esta última foi feita para representar subseções de uma *tag* <section>. Com isso, é possível inserir várias *tags* do tipo <article> dentro da *tag* <section>. Contudo, o contrário (*tags* <section> dentro de <article>) não faz sentido. O framework evita que este e outros casos sejam reproduzidos, bem como possibilita a melhora na navegação com leitores de tela, contribuindo para a melhoria da acessibilidade de deficientes visuais.

Os objetos que representam as *tags* possuem o seguinte formato base: **{ *type*: ..., *content*: ... }**. *Type* é um valor inteiro e define qual *tag* deve ser criada (é obrigatório). *Content* possui o que é necessário para criar a *tag*; dependendo de qual *tag* for criada, pode ser um objeto com suas próprias propriedades, uma *string*, ou uma lista (também é obrigatório).

Para tornar a utilização mais fácil e simplificada, o desenvolvedor não precisa criar cada uma dessas estruturas manualmente. Para isso, existem as funções auxiliares - uma função para cada *tag* implementada no *framework*, que recebe como parâmetros os valores necessários para a construção dos objetos. Cada uma delas possui o mesmo nome de sua respectiva *tag*. Elas podem ser encontradas na documentação do *framework* através do endereço do repositório apresentado anteriormente.

Para o *framework* funcionar é necessário realizar os seguintes passos:

1. Criar uma página HTML, e dentro da *tag* <header> importar o **jQuery**, o arquivo **framework.js** e o arquivo **framework.css**.

2. Criar um arquivo JavaScript separado e também o importar dentro da *tag* <header> da página HTML criada (é importante que o arquivo seja importado após o framwork.js, para que seja possível acessar as variáveis e funções auxiliares do *framework*).
3. Dentro do arquivo JavaScript criado no passo anterior é necessário implementar uma das seguintes funções: ***getHeaderObjects***, ***getMainObjects*** e ***getFooterObjects***. Todas elas devem retornar uma lista de objetos dos tipos que serão transformados nas *tags*. Cada uma será explicada detalhadamente mais a frente.
4. Adicionalmente, também é possível implementar outras duas funções: ***didStartSetup***, que é chamada logo após a página terminar de ser criada e antes do framework começar a chamar as funções implementadas no passo anterior para começar a construção da página; e ***didEndSetup***, que é chamada logo após o processamento de todos os objetos retornados nas funções anteriores. Ambas não são obrigatórias, não possuem retorno e podem ser usadas pelo desenvolvedor para executar alguma ação específica.
5. Também há a possibilidade de implementar uma função chamada ***getMenuColors***, que é utilizada para personalizar a aparência de um dos componentes implementados no *framework*. Ela também será explicada mais a frente.

Junto aos arquivos do *framework* no repositório, também há uma pasta chamada *template* que possui uma configuração básica do que foi dito para que o desenvolvedor possa ter um ponto de partida para começar a desenvolver.

Devido à limitação de tempo e a vasta quantidade de *tags* HTML existentes na linguagem, foi possível dar suporte para apenas uma parte delas nessa primeira versão do *framework*.

Um *layout* muito comum em páginas *web* é o que apresenta um conteúdo de cabeçalho no topo, o conteúdo principal da página no centro e um rodapé ao final. Devido a isso, o HTML5 possui as *tags* semânticas <header>, <main> e <footer>. Pensando em facilitar o trabalho dos desenvolvedores, as funções ***getHeaderObjects***, ***getMainObjects*** e ***getFooterObjects***, já inserem as *tags* criadas a partir dos objetos retornados, dentro das *tags* <header>, <main> e <footer> da página, respectivamente.

Além da estrutura padrão dos objetos que representam as *tags* no *framewok*, o desenvolvedor também pode adicionar qualquer outra propriedade ao objeto que foi criado, por exemplo, adicionar um identificador ou uma classe à uma *tag* específica, a fim de formatá-la posteriormente, ou utilizar CSS *inline* (formatar uma *tag* através de sua propriedade *style*) em alguma *tag*. Veja o exemplo nas figuras 1, 2 e 3.

```
function getMainObjects() {
    const link = a("Meu link personalizado", "#")
    link.id = "id-link-personalizado"
    link.class = "class-link-personalizado"
    link.style = "color: red; background-color: green;"
    link.target = "_blank"
    return [link]
}
```

Figura 1. Exemplo da construção da função getMainObjects() para formatação de um *link*.

```
▼ <main>
    <a href="#" id="id-link-personalizado" class="class-link-
    personalizado" style="color: red; background-color: green;" target=
    "_blank">Meu link personalizado</a>
  </main>
```

Figura 2. Código HTML gerado com a função getMainObjects().

Meu link personalizado

Figura 3. Saída na tela do navegador.

Ao desenvolver uma página *web* com formulário, o posicionamento dos *inputs* é uma tarefa trabalhosa para o desenvolvedor. Isso fica um pouco mais complicado quando é necessário levar em consideração a acessibilidade para deficientes visuais, pois os desenvolvedores devem tomar cuidado para que a formatação de estilo dos *inputs* não altere a ordem dos elementos na tela com relação a posição que eles estão no HTML (através do CSS é possível fazer isso), porque isso faz com que a navegação fique confusa para um usuário que navegue no formulário com uso do teclado.

Para evitar isso, o *framework* tem a propriedade *grid*. Uma propriedade de objetos que geram *tags* de formulário. Seu valor deve ser uma matriz de inteiros cujos elementos devem ter os valores de 0 à quantidade total de inputs do formulário. Cada valor da matriz representa um *input*, com isso tem-se que: o primeiro *input* é representado pelo valor 0, o segundo pelo valor 1, o terceiro pelo valor 2, e assim por diante. Cada linha da matriz representa uma nova linha no *layout* do formulário. Cada valor da linha representa a posição

do input correspondente ao valor, e uma nova coluna na respectiva linha na qual o valor se encontra.

Por exemplo, caso haja um formulário com 5 *inputs* e o valor da propriedade *grid* desse formulário seja passada como [[0, 1], 2, [3, 4]], isso significa que os dois primeiros *inputs* ficarão na primeira linha, o terceiro *input* na segunda linha, e o quarto e o último *inputs* na terceira linha - note que, quando uma linha possui somente um valor é possível omitir os colchetes (“[ ]”), como no caso do terceiro *input*. É possível ver um exemplo de implementação nas figuras 4, 5 e 6.

```
function getMainObjects() {
    const inputEmail = emailInput('email', 'E-mail: ', 'enter e-mail')
    const inputPass = passwordInput('password', 'Senha: ', 'enter password')
    const forgotPassword = a('Recover Password', 'www.fakehost.com/recover-pass')
    const enter = submit('Enter')
    const clear = reset('Clear')
    const formInputs = [inputEmail, inputPass, forgotPassword, enter, clear]
    const loginForm = form('www.fakehost.com/login', 'POST', formInputs)
    loginForm.style = 'width: 300px;'
    loginForm.grid = [0, 1, 2, [3, 4]]
    return [loginForm]
}
```

Figura 4. Exemplo de implementação da função getMainObjects() para construção de um formulário com o posicionamento dos *inputs* feito com a propriedade *grid*.

```
▼ <main>
  ▼ <form action="www.fakehost.com/login" method="POST" id="form-0" style="width:
  300px;">
    ▼ <label class="input-label" id="form-content-0" style="display: flex; flex-
    direction: row; justify-content: flex-start;">
        <span>E-mail: </span>
        <input name="email" type="email" placeholder="enter e-mail" class="input-
        small">
      </label>
    ▼ <label class="input-label" id="form-content-1" style="display: flex; flex-
    direction: row; justify-content: flex-start;">
        <span>Senha: </span>
        <input name="password" type="password" placeholder="enter password"
        class="input-small">
      </label>
      <a href="www.fakehost.com/recover-pass" id="form-content-2">Recover
      Password</a>
      <input type="submit" value="Enter" class="button-small" id="form-content-
      3">
      <input type="reset" value="Clear" class="button-small" id="form-content-4">
    </form>
  </main>
```

Figura 5. Código HTML gerado com a função getMainObjects(). Note que, cada *tag* de *input* é envolta por uma *tag* do tipo *label*. Isso foi gerado automaticamente pelo *framework* e é importante para que as TAs, como os leitores de tela, consigam identificar corretamente os *inputs* para os usuários.

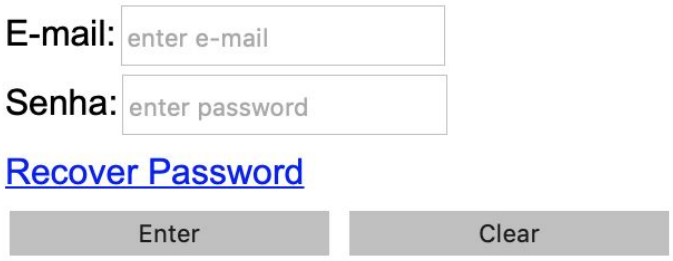

Figura 6. Saída na tela do navegador.

É possível ver no código HTML gerado e mostrado na figura 5 que as *tags* <form> e todas as outras *tags* aninhadas possuem *id*, mas que no código da figura 4 elas não foram especificadas. Isso ocorre porque para posicionar os elementos é necessário identificá-las, para que seja possível manipulá-las via CSS. Assim, caso o desenvolvedor não especifique tal propriedade, ela é automaticamente gerada.

Para que o posicionamento automático dos *inputs* de um formulário através do uso da propriedade *grid* funcione, as seguintes condições devem ser seguidas:

- O valor da propriedade deve ser uma matriz de inteiros.
- A matriz deve possuir todos os valores entre 0 e a quantidade total de *inputs* do formulário menos 1 (se o formulário possuir 10 *inputs*, a matriz *grid* deve possuir todos os valores de 0 a 9)
- Não deve existir elemento com valor menor que 0 ou maior que a quantidade total de *inputs* menos 1.
- Os elementos devem estar em ordem crescente. Isso se faz necessário para que não seja possível alterar a ordem dos componentes na tela com relação a ordem das *tags* no HTML, garantindo que usuários de teclado e leitores de tela tenham uma experiência adequada.

Caso as regras acima não sejam seguidas o *framework* não consegue posicionar os componentes e é exibido uma mensagem de erro no console do navegador.

O *framework* também disponibiliza um componente para construção de menus personalizados com acessibilidade. Ele é construído, assim como as outras *tags,* e sua função auxiliar chama-se ***menu*** (também existe a função encurtada ***mn***). Ela deve receber como parâmetro uma lista de objetos do tipo item de menu, que podem ser criados com a utilização da função ***menuItem*** (ou de sua versão encurtada ***mi***). Esta, por sua vez, possui os parâmetros *text* (nome do item de menu; deve ser uma *string* e é obrigatório), *link* (endereço para onde o usuário deve ser redirecionado quando pressionar esta opção; deve

ser uma *string* e também é obrigatório), *icon* (ícone que pode ser exibido na opção de menu; deve ser uma string e não é obrigatório) e *subitems* (sub opções da opção de menu; deve ser uma lista de objetos de item de menu e também não é obrigatório). Ver figuras 7 e 8.

É possível ver na figura 7 que são inseridas algumas propriedades adicionais ao objeto do menu. São elas:

- ***size***: usado para definir a altura do menu; pode receber um dos valores presentes nas variáveis *menu_small* (valor padrão), *menu_medium*, *menu_large*, *menu_extra_large* e *menu_extra_extra_large*.
- ***suboptions_icon***: usado para definir se deve ser exibido ou não o ícone de seta que indica que uma opção tem sub opções; pode receber os valore *true* ou *false* (valor padrão).
- ***icon_side***: usado para definir a posição dos ícones das opções do menu; pode receber um dos valores presentes nas variáveis *left* (valor padrão) e *right*.

O terceiro parâmetro passado na chamada das funções de criação dos objetos de itens do menu é utilizado para a criação do ícone da opção. Esse valor é adicionado com a propriedade *class* de uma *tag* <i>, e a biblioteca *Font Awesome* a transforma no ícone correspondente à classe.

```
function getMainObjects() {
    const menuItem1 = mi('Option 1', '#', 'fa fa-smile-o')
    const menuItem2 = mi('Option 2', '#', 'fa fa-smile-o')
    const menuItem3_1 = menuItem('Option 3.1', '#', 'fa fa-smile-o')
    const menuItem3_2 = menuItem('Option 3.2', '#', 'fa fa-smile-o')
    const menuItem3_3 = menuItem('Option 3.3', '#', 'fa fa-smile-o')
    const subitems = [menuItem3_1, menuItem3_2, menuItem3_3]
    const menuItem3 = menuItem('Option 3', '#', 'fa fa-smile-o', subitems)
    const simpleMenu = mn([menuItem1, menuItem2, menuItem3])
    simpleMenu.size = menu_extra_large
    simpleMenu.suboptions_icon = true
    simpleMenu.icon_side = right
    return [simpleMenu]
}
```

Figura 7. Exemplo de implementação da função getMainObjects() para construção de um menu de navegação.

Figura 8. Saída na tela do navegador.

O componente menu criado segue as especificações do padrão WAI-ARIA quanto a navegação e utilização da propriedade que indica se uma opção de menu está expandida ou não, possibilitando que usuários que utilizam o teclado possam navegar utilizando as teclas de seta e a tecla *tab*. O componente também é responsivo, e quando é exibido em telas menores que 800 *pixels* ele é transformado em uma lista com as opções uma abaixo da outra, e um botão que possibilita exibir o menu ou não é inserido logo acima da primeira opção - essa experiência é muito mais adequada para um dispositivo móvel, por exemplo. Ver figuras 9 e 10.

Figura 9. Exemplo de visualização de menu de navegação criado pelo *framework* em uma tela menor que 800 *pixels*. As opções do menu estão minimizadas e é possível abri-las ao pressionar o botão.

Figura 10. Exemplo de visualização de menu de navegação criado pelo *framework* em uma tela menor que 800 *pixels*. As opções do menu estão abertas e é possível minimiza-las ao pressionar o botão.

Para mudar as cores do menu, o desenvolvedor pode implementar a função ***getMenuColors***, que deve retornar uma lista de *strings* com as cores a serem usadas. Ver figuras 11 e 12.

```
function getMenuColors() {
    const backgroundColor = 'rgb(132,22,89)'
    const textColor = 'white'
    const backgroundColorHover  = 'rgb(112,10,69)'
    const textColorHover = textColor
    const sbmnBackgroundColor = backgroundColor
    const sbmnTextColor = textColor
    const sbmnBackgroundColorHover = backgroundColorHover
    const sbmnTextColorHover = textColor
    const colorsForMenu = menuColors([ backgroundColor, textColor,
                                       backgroundColorHover, textColorHover,
                                       sbmnBackgroundColor, sbmnTextColor,
                                       sbmnBackgroundColorHover, sbmnTextColorHover ])
    return [colorsForMenu]
}
```

Figura 11. Exemplo de implementação da função getMenuColors() para personalizar as cores de um menu de navegação criado pelo *framework*.

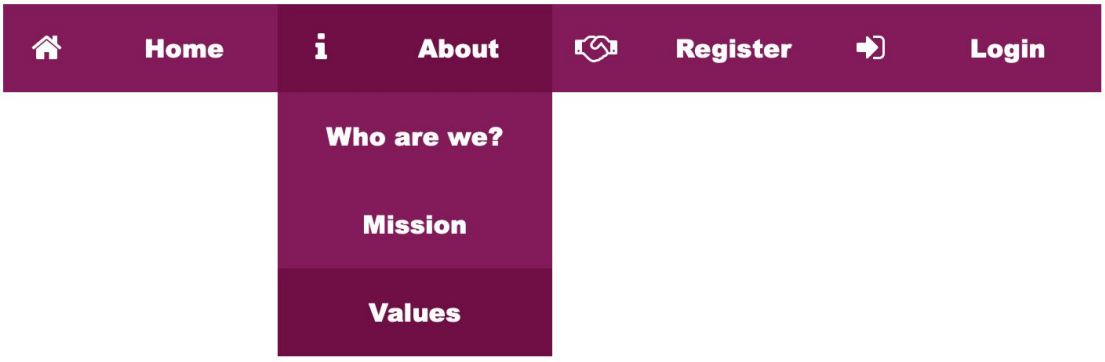

Figura 12. Exemplo de visualização de um menu de navegação criado pelo *framework* com cores personalizadas a partir da implementação da função getMenuColors().

O retorno da função deve possuir 8 valores, sendo os quatro primeiros para configurar a cor do menu, a cor do texto, a cor dos sub menus e a cor do texto dos sub menus, respectivamente; e as quatro últimas também configuram as mesmas propriedades de estilo, somente quando o cursor estiver em cima da opção. No caso do exemplo, as cores dos sub menus são iguais às do menu principal.

A função *getMenuColors* é chamada logo no inicio da execução do framework, e caso retorne valor, são criadas marcações de estilo global que serão aplicadas a todos os componentes do tipo menu da página.

A fim de verificar as contribuições do *framework* com relação a acessíbilidade para deficientes visuais das páginas *web* criadas com ele, foram realizados testes de navegação com o teclado e leitor de tela VoiceOver no componente de menu e em formulários.

O componente de menu segue as especificações do WAI-ARIA, sendo possível navegar entre os itens com a utilização das teclas de seta e da tecla *tab*, selecionar uma opção ou sub opção com a tecla *enter*; abrir um sub menu com a tecla de seta para baixo, navegar entre os componentes de um sub menu com uso das teclas de seta para cima e seta para baixo e fechar um sub menu com uso da tecla *esc*. As marcações do WAI-ARIA também tornam possíveis para usuários de leitores de tela, saberem quando uma opção que possui um sub menu está expandida ou não.

Já nos formulários, a propriedade *grid* possibilita posicionar os *inputs* sem apresentar os mesmos em uma ordem diferente da ordem do HTML, contribuindo para que usuários de teclado e leitores de tela tenham uma experiência adequada ao navegar pelo formulário, dando foco em um *input* por vez, um após o outro, sem pular algum componente ou voltar para trás (somente quando o usuário quiser voltar na navegação, assim como deve ser). Além disso, a inserção feita pelo *framework* da *tag* <label> nos *inputs* também melhora a acessibilidade, fazendo com que os leitores de tela leiam a identificação dos *inputs*.

5. CONSIDERAÇÕES FINAIS

O *framework* criado apresenta algumas facilidades de desenvolvimento de páginas *web* acessíveis para deficientes visuais, levando em consideração a navegação por leitores de tela e por teclado. Além disso, apresenta o componente de menu acessível com utilização das marcações do WAI-ARIA e *layout* responsivo, contribuindo também para acessibilidade das páginas *web*. A propriedade *grid* criada para os formulários também é um mecanismo que contribui para acessibilidade e responsividade, fazendo com que os tamanhos de seus componentes sejam ajustados de acordo com o tamanho da tela, mantendo as posições definidas pelo desenvolvedor, ao mesmo tempo que impossibilita a troca da ordem dos componentes apresentados na tela com relação a ordem do HTML.

Apesar disso, ainda existe muito trabalho a ser feito, deixando em aberto as seguintes possibilidades de trabalhos futuros:

- Testes com desenvolvedores e análise de desempenho para validação da ferramenta e verificação dos aspectos agilidade, responsividade e acessibilidade.
- Implementação das outras *tags* do HTML que não foram implementadas.
- Criação de mais opções de formatação.
- Implementação de mais componentes personalizados como o *menu*.

REFERÊNCIAS

ASSISTIVA. *O que é tecnologia assistiva?* 2019. Disponível em http://www.assistiva.com.br/tassistiva.html. Acesso em: 01 ago. 2019.

BRASIL. *eMAG Modelo de Acessibilidade em Governo Eletrônico*. Brasília: Ministério do Planejamento, Orçamento e Gestão, Secretaria de Logística e Tecnologia da Informação, 2014. Disponível em: https://www.governodigital.gov.br/documentos-e-arquivos/eMAGv31.pdf. Acesso em: 12 ago. 2019.

BRASIL. LEI N. 10.098, DE 19 DE DEZEMBRO DE 2000. *Estabelece normas gerais e critérios básicos para a promoção da acessibilidade das pessoas portadoras de deficiência ou com mobilidade reduzida, e dá outras providências*. Brasília, DF, dez 2000. Disponível em: http://www.planalto.gov.br/ccivil_03/leis/l10098.htm. Acesso em: 01 ago. 2019.

BRASIL. LEI No 13.146, DE 6 DE JULHO DE 2015. *Institui a Lei Brasileira de Inclusão da Pessoa com Deficiência (Estatuto da Pessoa com Deficiência)*. Brasília, DF, jul 2015. Disponível em: http://www.planalto.gov.br/ccivil_03/_ato2015-2018/2015/lei/l13146.htm. Acesso em: 01 ago. 2019.

BRASIL. *Relatório mundial sobre a deficiência*. São Paulo: SEDPcD, 2012. Disponível em: https://apps.who.int/iris/bitstream/handle/10665/44575/9788564047020_por.pdf?sequence=4. Acesso em: 01 ago. 2019.

CARVER, Metthew. *The Responsive Web*. New York: Manning Publications. 2014.

FUNDAÇÃO GETÚLIO VARGAS. *29ª pesquisa anual do uso de TI*. 2018. Disponível em https://eaesp.fgv.br/sites/eaesp.fgv.br/files/pesti2018gvciappt.pdf. Acesso em: 01 ago. 2019.

HENRY, Shawn Lawton; ABOU-ZAHRA, Shadi; BREWER, Judy. *The role of accessibility in a universal web. In Proceedings of the 11th Web for All Conference* (W4A '14). ACM, New York, NY, USA, Article 17, 4 pages, 2014. DOI=http://dx.doi.org/10.1145/2596695.2596719.

IBGE. Censo Demográfico. 2010. Disponível em: https://www.ibge.gov.br/estatisticas/sociais/populacao/9662-censo-demografico-2010.html. Acesso em: 01 ago. 2019.

JUNIOR, Jorge Fiore de Oliveira. *Avaliação de acessibilidade de software leitores de tela por pessoas com deficiência visual total com base nas diretrizes de acessibilidade para agente de usuário*. 2013, Monografia (Graduação em Sistemas da Informação) - Universidade Federal do Estado do Rio de Janeiro, Rio de Janeiro, 2013.

MAHMUD, Pablo Bizzi. *Um framework para apoiar o desenvolvimento de aplicações on-line acessíveis*. 2016. Dissertação de Pós-Graduação (Mestrado em Ciência da Computação).

TURRETCSS. turretcss - A Responsive Front-end Framework for Accessible and Semantic Websites. 2019. Disponível em: https://turretcss.com/. Acesso em: 10 ago. 2019.

W3C. Accessible Rich Internet Applications (WAI-ARIA) 1.1. 2017a. Disponível em: https://www.w3.org/TR/wai-aria/. Acesso em: 12. ago. 2019.

W3C. How WCAG 2.0 Differs from WCAG 1.0. 2009. Disponível em: https://www.w3.org/WAI/WCAG20/from10/diff.php. Acesso em: 10 ago. 2019.

W3C. HTML 5.2. 2017b. Disponível em: https://www.w3.org/TR/html52. Acesso em: 15 ago. 2019.

W3C. Web Accessibility Initiative (WAI). 2018a. Disponível em: https://www.w3.org/WAI/. Acesso em: 01 ago. 2019.

W3C. Web Accessibility Laws and Policies. 2018b. Disponível em: https://www.w3.org/WAI/Policy/. Acesso em: 01 ago. 2019.

W3C. Web Content Accessibility Guidelines (WCAG) 1.0. 1999. Disponível em: https://www.w3.org/TR/WAI-WEBCONTENT/. Acesso em: 10 ago. 2019.

W3C. Web Content Accessibility Guidelines (WCAG) 2.0. 2008. Disponível em: https://www.w3.org/TR/WCAG20/. Acesso em: 11 ago. 2019.

W3C. Web Content Accessibility Guidelines (WCAG) 2.*1*. 2018c. Disponível em: https://www.w3.org/TR/WCAG21/. Acesso em: 11 ago. 2019.

W3C. Design and Develop Overview. 2018d. Disponível em: https://www.w3.org/WAI/design-develop/. Acesso em: 11 ago. 2019.

WebAIM. WebAIM's WCAG 2 Checklist. 2018. Disponível em: https://webaim.org/standards/wcag/checklist. Acesso em: 11 ago. 2019.

WE ARE SOCIAL. Digital in 2018: world's internet users pass the 4 billion mark. 2018. Disponível em https://wearesocial.com/blog/2018/01/global-digital-report-2018. Acesso em: 01 ago. 2019.

WORLD HEALTH ORGANIZATION. *Disability and health*. 2018a. Disponível em: http://www.who.int/mediacentre/factsheets/fs352/en/. Acesso em: 01 ago. 2019.

WORLD HEALTH ORGANIZATION. *Vision impairment and blindness.* 2018b. Disponível em: http://www.who.int/mediacentre/factsheets/fs282/en/. Acesso em: 01 ago. 2019.

Contatos: renan.rsg@hotmail.com e mamelia@mackenzie.br