UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CÂMPUS GUARAPUAVA
CURSO DE TECNOLOGIA EM SISTEMAS PARA INTERNET

LUCIAN ROSSONI RIBAS

IMAGEJS: PLATAFORMA JAVASCRIPT PARA COMPOSIÇÃO DE UM MINIEDITOR DE IMAGENS

TRABALHO DE CONCLUSÃO DE CURSO

GUARAPUAVA
2017

LUCIAN ROSSONI RIBAS

IMAGEJS: PLATAFORMA JAVASCRIPT PARA COMPOSIÇÃO DE UM MINIEDITOR DE IMAGENS

Monografia de Trabalho de Conclusão de Curso de graduação, apresentada na disciplina de Trabalho de Conclusão de Curso 2 do Curso Superior de Tecnologia em Sistemas para a Internet - TSI da Universidade Tecnológica Federal do Paraná - UTFPR - Câmpus Guarapuava, como requisito parcial para obtenção do título de Tecnólogo em Sistemas para a Internet.

Orientador: Prof. Me. Guilherme da Costa Silva

GUARAPUAVA
2017

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Guarapuava
Curso Superior de Tecnologia em Sistemas Para a Internet

ATA DE DEFESA DE MONOGRAFIA DE TRABALHO DE CONCLUSÃO DE CURSO DO CURSO DE TSI

No dia 21 de junho de 2017, às 16:30 horas, nas dependências da Universidade Tecnológica Federal do Paraná Câmpus Guarapuava, ocorreu a banca de **defesa da monografia** de Trabalho de Conclusão de Curso intitulada: "**IMAGEJS: Plataforma Javascript para Composição de um Mini Editor de Imagens**" do acadêmico **Lucian Rossoni Ribas** sob orientação do professor **Prof. Me. Guilherme da Costa Silva** do Curso de Tecnologia em Sistemas para Internet.

Banca Avaliadora	
Membro	**Nome**
Orientador	Prof. Me. Guilherme da Costa Silva
Coorientador	Prof. Me. Emerson Andre Fedechen
Avaliador 1	Prof. Dr. Roni Fabio Banaszewski
Avaliador 2	Prof. Me. Diego Marczal

Situação do Trabalho	
Situação	(x) Aprovado ( ) Aprovado com ressalvas ( ) Reprovado ( ) Não Compareceu
Encaminhamento do trabalho para biblioteca	(x) Pode ser encaminhado para biblioteca. ( ) Manter sigilo para publicação ou geração de patente.

Guarapuava, 21 de junho de 2017.

A Folha de Aprovação assinada encontra-se na Coordenação do Curso (ou Programa).

AGRADECIMENTOS

Agradeço principalmente ao meu irmão Yurick Vinicius Ribas que por seu incentivo ingressei no curso referente. Conjuntamente, neste espaço que seguramente sempre estarão, dedico enorme gratidão em função do excelente suporte familiar que recebi durante o tempo acadêmico, meu pai Sebastião Ribas Filho e minha mãe Rosemari Rossoni Ribas.

Ao acadêmico Darlan Lara, deixo minhas gratificações por desenvolver voluntariamente o ícone representativo da plataforma proposta, deixo também meu reconhecimento pelo excelente trabalho de *design* empregado.

Agradeço a acadêmica de Fisioterapia Maiara Fonseca, e ao Dr. Rodrigo Barbosa e Silva pela prontificação arbitrária em ajudar-me a enriquecer este documento de maneira mais conclusiva e descritiva.

Por fim, agradeço a todos os integrantes da comunidade acadêmica, desde os funcionários e docentes que exercem sua profissão com entusiasmo aos alunos que possibilitam a existência do câmpus; carinhosamente àqueles que participaram da minha formação. Especificamente, agradeço ao professor Guilherme da Costa Silva por em último prazo assumir a orientação do trabalho e pelo suporte conceitual fornecido durante o desenvolver deste documento.

Lucian Rossoni Ribas

If you have capacity, achievements and celebrations are trivial; do not take advantage of specific situations to demonstrate it, demonstrate it all the time. (The Author, 2017)

Se possui capacidade, conquistas e celebrações são triviais; não tire proveito de situações específicas para demonstrá-la, demonstre-a o tempo todo. (O Autor, 2017)

RESUMO

RIBAS, Lucian. IMAGEJS: PLATAFORMA JAVASCRIPT PARA COMPOSIÇÃO DE UM MINIEDITOR DE IMAGENS. 86 f. Trabalho de Conclusão de Curso – Curso Superior de Tecnologia em Sistemas para Internet, Universidade Tecnológica Federal do Paraná. Guarapuava, 2017.

Atualmente, as principais plataformas desenvolvidas para edição de imagens ainda são predominantemente direcionadas para dispositivos *desktop*; porém, o aumento da capacidade computacional direcionada em recursos para Internet está desencadeando, lentamente, a migração desses programas para o ambiente Web, incluindo os *softwares* desenvolvidos para manipulação de imagens, definindo o escopo desse projeto. Disponibilizado no formato de um mini estúdio com bibliotecas e dependências JavaScript, esta plataforma tem como objetivo incorporar em uma página Web um pequeno editor para manipulação de imagens, agrupando de forma gráfica algumas ferramentas que permitem que o usuário aplique manipulações básicas em uma imagem. A imagem poderá ser selecionada como arquivo, pela árvore de diretórios locais, ou enviada a partir de um endereço remoto externo. Por se tratar de uma biblioteca com intuito *cross-browser*, ela será de grande utilidade para desenvolvedores de sistemas para Web, sendo que estes não precisarão implementar módulos referentes a edições básicas de imagens.

Palavras-Chave: Estúdio de Edição. Manipulação de Imagens. Desenvolvimento Web. *Framework* JavaScript.

ABSTRACT

RIBAS, Lucian. IMAGEJS: JAVASCRIPT PLATFORM FOR COMPOSITION OF A MINI IMAGE EDITOR. 86 f. Trabalho de Conclusão de Curso – Curso Superior de Tecnologia em Sistemas para Internet, Universidade Tecnológica Federal do Paraná. Guarapuava, 2017.

Currently, the major platforms developed for image editing are still predominantly targeted to desktop devices; however, the increase in computational capacity directed at resources for the Internet is slowly triggering the migration of these programs to the Web environment, this includes softwares developed for image manipulation, defining the scope of this project. Available in the format of a mini studio with JavaScript libraries and dependencies, this platform aims to incorporate in a Web page a small editor for manipulation of images, graphically grouping some tools that allow the user to apply basic manipulations to an image. The image can be selected as a file, by the local directories tree, or sent from an external remote address. Due to the intention of being a cross-browser library, it will be very useful for Web-based system developers, who will not need to implement modules related to basic image editions.

Keywords: Editing Studio. Image Manipulation. Web Development. Framework JavaScript.

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE SIGLAS

HTML	Linguagem para Marcação de Hipertexto (do inglês *HyperText Markup Language*)
CSS	Folhas de Estilo em Cascata (do inglês *Cascading Style Sheets*)
JS	JavaScript
DOM	Modelo de Objeto de Documentos (do inglês *Document Object Model)*
WWW	Rede mundial de computadores (do inglês *World Wide Web*)
API	Interface de Programação de Aplicativos (do inglês *Application Program Interface*)
ECMA	Associação Europeia de Fabricantes de Computadores (do inglês *Europe Computer Manufacturers Association*)
INPE	Instituto Nacional de Pesquisas Espaciais
II	Índice de Indentação referente no apêndice do contexto em questão
IE	Internet Explorer
POO	Programação Orientada a Objetos
F/R	Formato/Retorno; Formato ou Retorno
RGB	Vermelho, verde, azul (do inglês: *Red*, *Green, Blue*)
URL	Localizador Uniforme de Recursos (do inglês *Uniform Resource Locator*)
HTTP	Protocolo de Transferência de Hipertexto (do inglês *Hypertext Transfer Protocol*)

SUMÁRIO

1 INTRODUÇÃO

Atualmente é comum que pessoas com um computador em mãos elaborem edições em imagens, vídeos, fotografias, filmes, entre outros dados no formato de mídias digitais. A necessidade da manipulação gráfica deste tipo de dados tornou-se responsabilidade da computação presente no dia-a-dia; ou seja, recursos simples, de fácil acesso, utilização e principalmente, que atendam as necessidades mínimas do usuário habitual.

Devido essa necessidade de edição, existem diversos recursos computacionais desenvolvidos para prover soluções a estes tipos de recursos. Muitos *softwares* são construídos e disponibilizados para praticamente todos os ambientes de acessibilidade, tais como: dispositivos móveis, computadores *desktop*, entre outras classes de aparelhos. A Internet, devido a sua abrangência, é um dos meios com grande abundância de aplicativos desse gênero; e a quantidade de tecnologias e usuários que acessam estes programas está crescendo juntamente com os recursos disponibilizados em nuvem, assim como afirma a matéria publicada por Marcel Gugoni, na página Amcham Brasil[1]; para Taurion (2009, p. 17), computação em nuvem significa: "Utilizar os recursos ociosos de computadores independentes, sem preocupação com localização física e sem investimentos em *hardware*."; ou em conceito simples segundo Santaella (2014): "Sugerir o armazenamento e processamento de dados e prestação de serviços de grandes complexos de dados através da Internet.".

O usuário corriqueiro está se adaptando ao uso de sistemas disponíveis na Internet para auxiliar nas atividades e interações virtuais, fazendo com que esse tipo de recurso torne-se comum no seu dia-a-dia. Por exemplo, segundo a matéria publicada em 2013 pela IdgNow[2], o Google (sistema para pesquisas *online*) recebe por dia aproximadamente 3 (três) bilhões de pesquisas, onde 15% destas são inéditas; assim como a matéria divulgada pelo

[1] A Amcham Brasil é uma associação de empresas e instituições brasileiras que visam soluções políticas e administrativas.
[2] A IdgNow é um portal de notícias tecnológicas.

jornal O Globo em sua página[3], informa que por ano, 125 bilhões de imagens publicadas somente em redes sociais.

Percebe-se que muitos dos sistemas presentes na Web[4] possuem alguma relação com arquivos no formato de mídias, seja para o envio de desenhos, imagens, fotos, entre outros. Devido a isso, seria interessante que estes sistemas possuíssem alguma etapa para aplicar pré-edições nas imagens que serão enviadas pelos usuários. Entretanto, muitos dos sistemas Web disponibilizam apenas a opção do envio de imagens, considerando que as edições já tenham sido previamente feitas pelo próprio usuário através de outros programas.

Em virtude da dificuldade de desenvolvimento ou até mesmo por questões financeiras, é trabalhoso encontrar recursos que sejam facilmente vinculáveis para a utilização por programadores de sistemas Web. Logo, a maioria das plataformas não possuem opções do gênero, pois para desenvolver tais ferramentas desde o início da sua implementação seria necessário muito mais tempo, conhecimento, e mão de obra tecnológica direcionada ao sistema.

Para poupar tempo de desenvolvimento, utilizam-se *frameworks* que têm como objetivo apoiar os desenvolvedores na construção e mantimento de determinados projetos, oferecendo uma série de códigos e funcionalidades prontas e testadas. Segundo Fayad e Schmidt (1997, p. 10), *"Frameworks* são um conjunto de classes pré-fabricadas de *software*, que trabalham em conjunto para realizar uma ou mais responsabilidades para um domínio de um subsistema da aplicação."

O escopo deste trabalho foi o desenvolvimento de uma plataforma de edição *client-side*[5] titulada pelo autor como ImageJS (referenciando imagem e JavaScript), que será utilizada através de Web navegadores. Não será possível identificar qual navegador será utilizado para acessar os sistemas que possuirão o minieditor vinculado; portanto, a plataforma é genérica neste contexto, condizente com um recurso *cross-browser ou* multi-navegador. – "A *Cross-Browser* compatibilidade entre navegadores é a prática de testar projetos em todos os principais sistemas de navegadores e é importante no *design* da Web porque você não apenas projeta sites para si mesmo, mas para todo um público na Web. [...]" (SABIN-WILSON, 2013, p.76).

---

[3] O Globo é um jornal online de notícias gerais.
[4] Termo britânico informal criado em 2004 pela pela empresa O'Reilly Media, referência a World Wide Web (WWW).
[5] A linguagem de cliente ou *client-side* é uma linguagem executada no lado cliente.

Para que uma plataforma possa ser considerada um estúdio ou editor de imagens, é necessário que possua ferramentas de controle das imagens, para aplicar as edições, e principalmente que detenha de uma interface dinâmica de interação para o usuário, caso contrário o recurso não passaria de uma biblioteca. O objetivo deste projeto não está apenas em projetar uma biblioteca ou um *framework*, mas um minieditor que permita que o usuário aplique as edições sem dificuldades, de maneira simples e objetiva.

Estruturar um minieditor é um procedimento bastante exigente, pois funcionará na mesma página de vinculação, como um componente, devido a isso deu-se prioridade neste procedimento, com objetivo de criar uma estrutura eficiente para comportar futuras ferramentas de edição. Muitas ferramentas que foram desenvolvidas fazem referências aos recursos do editor e não diretamente a uma funcionalidade de edição de imagem (tais como vincular e resetar a imagem que está sendo editada, inserir figura da Web, entre outros). A Tabela 1 lista todos os recursos que o editor dispõe até o momento.

Tabela 1 – Lista de Ferramentas Desenvolvidas

Ferramentas de Controle	**Ferramentas de Edição**
Abrir Editor	Aplicar Brilho
Fechar Editor	Aplicar Contraste
Editar Imagem do Dispositivo	Realçar Tom Vermelho
Editar Imagem da Web	Realçar Tom Verde
Editar em Modo de Tela Cheia	Realçar Tom Azul
Retornar Edição ao Estado Inicial	Filtro Roxo
Salvar Imagem Editada	Filtro Preto e Branco
	Recorte (Demonstrativo)
	Borragem (Demonstrativa)

1.1 OBJETIVOS

Nesta seção serão apresentados o objetivo geral e os objetivos específicos do presente trabalho. No objetivo geral será descrito de maneira sucinta e genérica o que se deseja atingir com o fim do desenvolvimento; nos objetivos específicos será descrito tudo que se deseja estudar e desenvolver para alcançar o objetivo geral.

1.1.1 OBJETIVO GERAL

O objetivo geral deste projeto consiste no desenvolvimento de uma plataforma de edição incorporável e aplicável em uma página Web, compondo um minieditor *client-side* para manipulações básicas em uma imagem.

1.1.2 OBJETIVOS ESPECÍFICOS

São objetivos específicos deste trabalho os seguintes itens:

1. Projetar e desenvolver um minieditor dinâmico e independente da plataforma de vinculação;
2. Definir princípios de autonomia HTML, CSS e JS na construção de Componentes;
3. Estudar o Desenvolvimento multi-navegador;
4. Aplicar princípios de implementação e desenvolvimento de recursos vinculáveis em navegadores Web (Componentes);
5. Aplicar técnicas básicas de processamento de imagem (que não manipulam os pixels vizinhos).

6. Projetar e desenvolver uma biblioteca JavaScript independente para controlar a dinâmica da plataforma;
7. Desenvolver uma página Web realizando a simulação de vinculação e utilização dos recursos propostos.

1.2 DIFERENCIAL TECNOLÓGICO

Um dos principais diferenciais desse projeto é a possibilidade de implementar de forma simples à incorporação em qualquer formulário de entrada de dados da Web. Onde, por exemplo, a única configuração necessária para que o usuário da página utilize os recursos oferecidos pelo ImageJS será inserir a *tag*[6] HTML de vinculação do editor dentro do código fonte da página (independente das tecnologias que já estejam envolvidas) e internamente ao formulário que será enviado ao servidor, conforme mostra o exemplo na Figura 1:

```
<!-- Um formulário de dados qualquer -->
<form>

    <!-- Tag singular de invocação do ImageJs -->
    <image-js></image-js>

    <!-- inputs de dados -->
```

Figura 1 – ImageJS, Exemplo de Vinculação do Minieditor ImageJS

Fonte: Autor.

Considera-se diferencial também o fato do recurso estar disponível ao usuário no formato de um minieditor para edição de imagens, e não apenas como bibliotecas independentes e estilos vinculáveis. Para melhor compreensão do que se trata um editor de imagens foi destacado um exemplo visual na Figura 2.

[6] Marcações. *Tags* são estruturas de linguagem de marcação contendo instruções, tendo uma marca de início e outra de fim para que o navegador possa renderizar uma página.

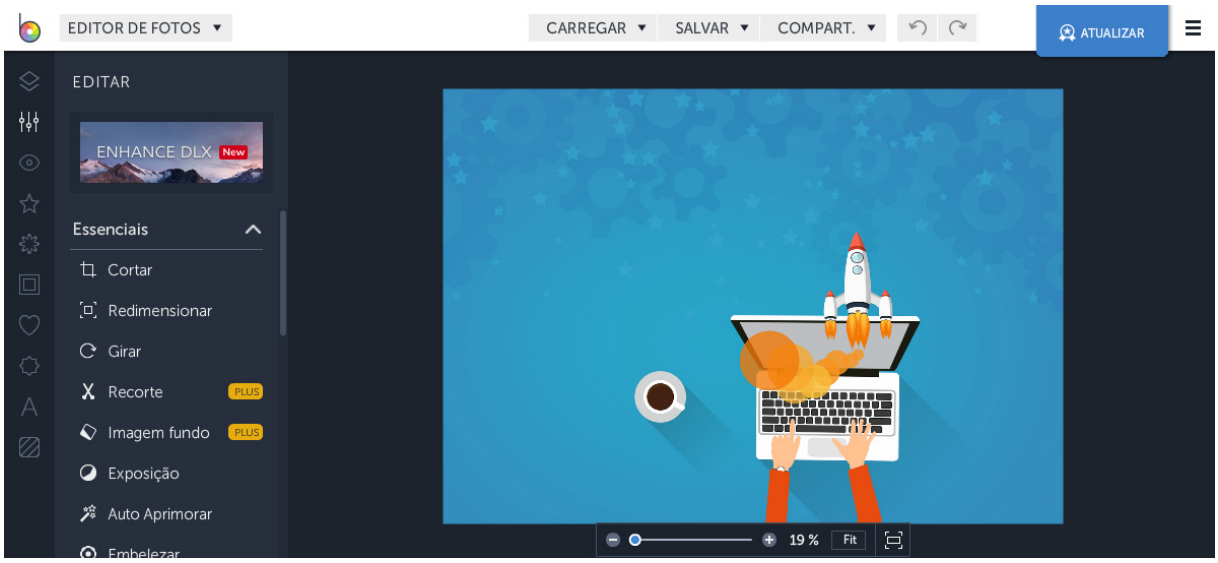

Figura 2 – Exemplo de Estúdio Editor de Imagens (Befunky)[7]

Fonte: Befunky, Experimente Já[8].

Outro diferencial é a obtenção de resultados utilizando o método de codificação de informações no formato Base64[9], possibilitando abrangência tecnológica aos serviços que irão receber a requisição de envio das edições (pois o envio será no formato textual); já que muitas linguagens de programação usadas em servidores possuem suporte a tal recurso.

Outras características que podem ser consideradas como diferenciais:

1. Interface leve e objetiva (poucos clicks se utilizar os recursos da ferramenta) baseada nos estilos Material (seção 2.5), que são princípios visuais *front-end*[10] desenvolvido pela Google.
2. Código aberto possibilitando que a comunidade interessada contribua no desenvolvimento da plataforma.
3. Todo seu código fonte de produção está localizado em um único arquivo que não ultrapassa os 50 kbytes de espaço em armazenamento físico.

[7] A imagem "Computador e Foguete" utilizada como exemplo de edição foi obtida gratuitamente pelo site Stockvault (<https://www.stockvault.net/>, acesso em 22/05/2017) e será utilizada como representativo de edição durante o decorrer do documento.
[8] <https://www.befunky.com/pt/criar/aprimorador-de-fotos/>. Acesso em 22/05/2017.
[9] *Base64* é uma maneira de codificarmos os dados para transferi-los na internet.
[10] *Front-end* é a parte visual de um sistema, área de interação e entrada de dados dos usuários.

4. As *tags* HTML (seção 5.1) e Folhas de Estilo em Cascata (CSS, seção 5.2) estão escritas de maneira a não afetar o sistema em que a plataforma está sendo vinculada.

1.3 ESTRUTURA DO TRABALHO

Esta monografia está dividida da seguinte forma: no Capítulo dois é descrito a fundamentação teórica do trabalho, bem como as tecnologias estudadas e quais foram utilizadas. No terceiro Capítulo é apresentado o estado a arte do projeto, descrevendo e comparando com outros trabalhos na mesma área de pesquisa. No Capítulo quatro é a análise e projeto do sistema seguida para desenvolver todas as etapas e planejamentos do projeto. O quinto Capítulo é sobre o desenvolvimento, descrevendo detalhadamente como o trabalho foi implementado. O sexto Capítulo apresenta como são feitos os testes na plataforma em diferentes escopos. No oitavo Capítulo, descreve-se o comportamento da plataforma em simulação com ambiente real. E no oitavo Capítulo são descritas as considerações finais sobre o projeto.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentadas as tecnologias e conceitos utilizados no desenvolvimento do sistema que constitui o projeto. Foram listados alguns princípios de interface, linguagens *front-end* de marcação e estilização, linguagens de programação e *frameworks* para o respectivo escopo.

2.1 O QUE É SISTEMA VINCULANTE

O termo Sistema Vinculante é utilizado repetidamente no desenvolver do documento referente. Trata-se de um conceito com significado interno ao projeto já que faz referência características que envolvem o trabalho proposto. Sistema Vinculante, singularmente falando representa o sistema que vinculará, no conceito do editor: o sistema recipiente, que terá em seu conteúdo a importação dos recursos do ImageJS, ou seja, uma página HTML com as bibliotecas e *tags* vinculadas.

2.2 O QUE É UMA PLATAFORMA

Segundo a empresa HSM Management (2017), não existe, em nenhum dicionário tradicional ou até mesmo no Google ou na Wikipedia, uma definição formal para esse termo, trata-se de um conceito ainda em gestação.

Por se tratar de um termo genérico, o conceito varia em função do escopo de sua aplicação. Em computação, segundo Joaquim Torres (2015), “Uma plataforma computacional é qualquer ambiente computacional onde um software será executado”. Ele também lista vários tipos de plataformas, tais como:

- **Plataforma de Troca:** Plataforma que reúne compradores e vendedores: MercadoLivre, Uber e Airbnb.
- **Plataforma de Conteúdo:** Plataforma onde o conteúdo é o foco, e a monetização é feita normalmente por meio de anúncios (Google, Facebook e portais de notícias).
- **Plataforma Tecnológica:** Onde a plataforma é o sistema operacional e, de um lado, temos usuários e, do outros, temos desenvolvedores, como: Linux, Windows, App Store e Android.

Tendo em vista que de um lado do editor proposto estão os desenvolvedores e contribuintes, do outro lado existem os usuário dos sistemas vinculantes; baseando-se nos conceitos listados, o projeto atual melhor se enquadra no conceito de Plataforma Tecnológica.

2.3 LINGUAGEM DE MARCAÇÃO DE HIPERTEXTO

Para compartilhar informações de maneira universal nas Web, é necessário haver um padrão que seja entendido por diversos meios de acesso (dispositivos), interpretada por navegadores Web, a Linguagem de Marcação de Hipertexto (do inglês *Hypertext Markup Language*) ou HTML, se propõe a ser a linguagem com este padrão. Desenvolvido originalmente por Tim Berners-Lee o HTML ganhou popularidade quando o Mosaic – *browser* desenvolvido por Marc Andressen na década de 1990 – ganhou força (W3C, 2010).

Um arquivo, documento ou página HTML pode ser descrito da seguinte maneira:

> *Uma página inicial, conhecida como home page ou página Web, é um arquivo textual comum e armazenado na extensão .HTM ou .HTML cujo o conteúdo é composto basicamente de textos e códigos especiais chamados tags que possibilitam a exibição destes arquivos na Web (WWW ou World Wide Web) através de um*

> *programa especial chamado navegador ou Agente de Usuário*[11]. (RAMALHO, 2005)

O HTML ganhou as versões HTML+, HTML 2.0 e HTML 3.0 entre 1993 e 1995, onde foram propostas várias mudanças para o enriquecimento da linguagem. Contudo, até então, o HTML não era tratado como um padrão no mundo da Web. Apenas em 1997, o grupo de trabalho da W3C responsável por manter o padrão do código, lançou a versão 3.2 da linguagem, fazendo com que ela se torne uma prática comum na Web (W3C, 2010).

Atualmente o HTML se encontra na sua versão 5 e seu o principal objetivo é o de facilitar a manipulação de elementos e objetos do DOM[12]. Ao contrário das versões antecessoras, o HTML5 disponibiliza ferramentas para que as CSS (seção 2.4) e o JavaScript (seção 2.6) da página cumpram com seu trabalho da melhor maneira possível. O HTML5 permite por meio das suas APIs[13] a manipulação das características destes elementos, de forma que a página Web continue leve e funcional (W3C, 2010).

O diferencial do HTML5 pode ser sucintamente descrito da seguinte maneira:

> *As versões antigas do HTML não continham um padrão universal para a criação de seções comuns e específicas como rodapé, cabeçalho, sidebar, menus e etc. Não havia um padrão de nomenclatura de ID, Classe (ID e Classe são atributos de um elemento HTML) ou tags. Não havia um método de capturar de maneira automática as informações localizadas nos rodapés dos websites. O HTML5 propôs uma forma mais organizada e semântica de efetuar a marcação nos textos* (W3C, 2010).

A *tag* Canvas do HTML surgir apenas após a versão 5 da linguagem, este recurso possui extrema importância lógica no projeto em questão, pois as ferramentas propostas foram desenvolvidas baseiam-se nas propriedades desse elemento. Segundo a API *online* da Mozilla [14], o elemento canvas pode ser descrito da seguinte maneira:

---

[11] Termo empregado para se fazer referência a qualquer dispositivo capaz de interpretar um documento escrito em linguagem de marcação. O exemplo mais comum e conhecido de agente de usuário é o navegador. Leitores de tela, robôs de indexação e busca, dispositivos móveis como tablets e smarthphones são também alguns exemplos de agentes de usuário. (SILVA, 2012)

[12] DOM: Modelo de Objeto de Documentos (do inglês *Document Object Model*)

[13] API: Interfaces de Programação de Aplicativos (do inglês *Application Program Interface*)

[14] <https://developer.mozilla.org/pt-BR/docs/Web/HTML/Canvas>. Acesso 27/05/2017.

> *Adicionado ao HTML5, o elemento HTML <canvas> é um elemento que pode ser usado para desenhar gráficos via código (normalmente JavaScript). Por exemplo, ele pode ser usado para desenhar gráficos, fazer composição de fotos, criar animações ou até mesmo fazer processamento ou renderização de vídeo em tempo real.* (Mozilla Foundation, 2014).

2.4 FOLHAS DE ESTILO EM CASCATA

As CSS, traduzido do português como Folhas de Estilo em Cascata (do inglês, *Cascading Style Sheets*), tem por finalidade aplicar uma marcação HTML/XML no estilo visual dos elementos. Por definição, não cabe ao HTML fornecer informações ao Agente do Usuário sobre a apresentação dos elementos, como por exemplo: cores de fontes, tamanhos de textos, posicionamento dos demais aspectos visuais de um documento. Cabem às CSS todas as funções de apresentação de um documento. Em setembro de 1994 surge a primeira proposta de implementação da CSS, mas apenas em 17 dezembro de 1996 as CSS foram lançadas como uma recomendação oficial do W3C, padronizando a sua primeira versão, as CSS1 (SILVA, 2012).

Atualmente, a CSS se encontram na sua versão 3, onde sua principal diferença é o modelo de desenvolvimento adotado. Enquanto as versões anteriores da CSS adotaram modelos de desenvolvimento baseados em um único documento, as CSS3 estão sendo desenvolvidas em módulos, permitindo o desenvolvimento independente de cada módulo, possibilitando que os fabricantes implementem funcionalidades previstas em módulos futuros, adiantando o desenvolvimento do Agente de Usuário em questão (SILVA, 2012).

O desenvolvimento das aplicações que usam CSS pode ser facilitado por ferramentas e regras *front-end* pré-definidas. Neste projeto, devido aos seus princípios robustos, bonitos e simples de usar, adotou-se os estilos de interface do Material Design, desenvolvido pela Google (seção 2.5).

Toda a interface do ImageJS foi desenvolvida pelo autor baseando-se nos princípios do Material Design (tópico a seguir); no entanto, a biblioteca de ícones desenvolvida e publicada gratuitamente pela Google, foi diretamente vinculada ao projeto, sem quaisquer edições.

2.5 MATERIAL DESIGN

Segundo a Google[15], empresa desenvolvedora dos estilos visuais Material Design, a ideia do projeto Material é proporcionar aos seus usuários uma experiência visual inovadora, com um padrão de linguagem de apresentação que se aplique aos princípios clássicos de um bom *design*, oferecendo inovação, tecnologia e possibilidade de aplicação de ciência (definição de cores e formatados mais aceitos pelo olho humano) nos aspectos dos elementos.

Devido à boa aceitação pelos usuários do modelo visual Material, a comunidade Web passou a desenvolver vários *frameworks front-end* que utilizam dos conceitos e modelos, como por exemplo o Materialize[16]

Para melhor entender uma página baseada nos estilos Material, observe a Figura 3, exemplificando sucintamente como aparenta os estilos em mais de um tipo de dispositivo.

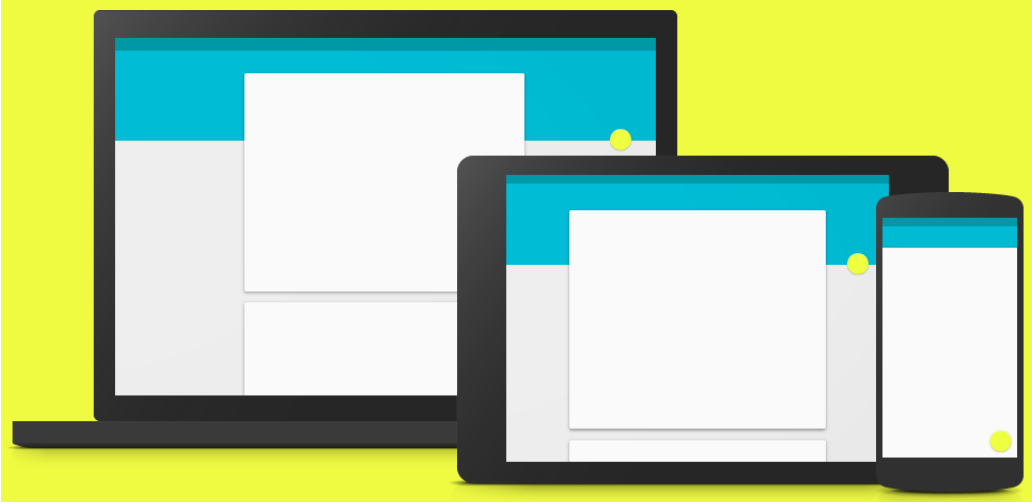

Figura 3 – Exemplos de Páginas Baseada no Material Design

Fonte: W3C, Material[17].

[15] Empresa multinacional de *softwares* dos Estados Unidos.
[16] Linguagem de *design* desenvolvida pela Google.
[17] <https://www.w3schools.com/w3css/w3css_material.asp>. Acesso em 22/05/2017.

2.6 LINGUAGEM DE PROGRAMAÇÃO JAVASCRIPT

A linguagem JavaScript foi criada pela Netscape[18], para controle de objetos XMLHttpRequest[19] do DOM. Ao início do projeto chamava-se LiveScript, uma linguagem baseada no C, no C++ e no Perl. Dado que no mesmo período foi introduzido a linguagem Java (REMOALDO, 2008).

A JavaScript pode ser definida brevemente da seguinte maneira:

> *A JavaScript é uma linguagem de uso global de descendência híbrida, com uma pequena semelhança com a família das linguagens C. A JavaScript, pode ser considerado basicamente como uma linguagem de criação de scripts de uso geral, interpretada e fracamente tipada. Fracamente tipada significa que as variáveis não são declaradas especificamente como strings, inteiros ou objetos e que é possível atribuir valores de diferentes tipos à mesma variável.* (CRANE, 2008)

Dado que no mesmo tempo foi introduzida a linguagem Java, e através de um acordo com a Sun, a Netscape resolveu mudar o nome da linguagem para JavaScript e disponibilizou a versão 1.0 em março de 1996 no Navigator 2.0 (REMOALDO, 2008).

Devido ao fato da Netscape não ter licenciado o JavaScript, em 1996 a Microsoft[20] lançou uma linguagem chamada JScript para atribuir ao Internet Explorer 3.0. As linguagens tinham suas semelhanças, mas seus scripts não eram iguais, gerando problemas de incompatibilidades nas páginas web, obrigando os desenvolvedores a implementar dois códigos diferentes quando seus usuários utilizam diferentes navegadores (REMOALDO, 2008).

Em 1997 foi lançada uma norma chamada ECMA-262 pela ECMA[21], que tem como objetivo incorporar os elementos tanto do JavaScript como do JScript em uma linguagem chamada ECMAScript (REMOALDO, 2008).

Assim como em outros escopos e tecnologias, o desenvolvimento das aplicações que usam JavaScript pode ser facilitado por bibliotecas, ferramentas e *frameworks* criados para

---

[18] Empresa de serviços de computadores dos Estados Unidos.
[19] XMLHttpRequest: Objeto JavaScript responsável pela comunicação assíncrona com um servidor web.
[20] Empresa transnacional de *softwares* e outros produtos computacionais dos Estados Unidos.
[21] ECMA: Associação Europeia de Fabricantes de Computadores (do inglês Europe Computer Manufacturers Association) – uma associação internacional de indústrias sem fins lucrativos

auxiliar o programador. Neste projeto, devido a variedade de códigos e funcionalidades implementadas pela comunidade será utilizado o *framework* conhecido como jQuery (seção 2.7), que é uma biblioteca do JavaScript padronizada nos princípios Facade[22].

2.7 A BIBLIOTECA JQUERY

A jQuery é uma biblioteca JavaScript desenvolvida para auxiliar Web *designers* e desenvolvedores Web a criar e ampliar as interações JavaScript de maneira rápida e concisa, usando um conjunto definido de métodos que envolvem as funções nativas do JavaScript. A biblioteca jQuery não oferece qualquer funcionalidade nova, ela usa as APIs existentes do JavaScript, que são cansativas e repetitivas, e as disponibiliza para um público mais amplo por meio de sua sintaxe de fácil compreensão e criação (RUTTER, 2012).

Devido a sua flexibilidade e sintaxe amigável, a jQuery acabou se tornando a biblioteca JavaScript mais utilizada pelos sistemas Web. Em 2008 passou a ser a biblioteca padrão do ASP.NET MVC, da Microsoft, e o Ruby on Rails, a partir da versão 3, substituiu o Prototype pelo jQuery (BALDUINO, 2012).

Atualmente o jQuery se encontra na versão 2 e sua parte fundamental ainda são os *selectors*. Os *selectors* indicam o quê será utilizado. Basicamente, se entender o conceito de *selectors*, entendeu a maior parte do funcionamento do *framework* (BALDUINO, 2012).

O jQuery possui uma biblioteca implementada para manipulação de imagem chamada jCanvas. Segundo seu site oficial se trata de uma biblioteca JavaScript, escrita usando jQuery e para jQuery, que envolve a API canvas do HTML5, adicionando novos recursos e capacidades, muitos dos quais são customizáveis. Os recursos incluem camadas, eventos, drag-and-drop, animação, entre outros.

[22] Facade: Padrão de projeto que propõe uma interface otimizadora de funcionalidade de uma API.

2.8 LINGUAGENS DE SERVIDOR, PHP E RUBY

Nos testes descritos posteriormente (seção 6.1), foram utilizadas duas linguagens de servidoras para receber os envios do formulário que o ImageJS estiver vinculado. Ambas as tecnologias não possuem nenhuma relação direta com a plataforma proposta (pois a plataforma é totalmente *client-side*), portanto esse tópico será breve.

O PHP e o Ruby foram escolhidos devido a popularidade de ambas no ambiente Web e pelo o entrosamento que o autor já possuía com as linguagens; porém, pode-se utilizar qualquer ambiente de servidor para receber o envio do sistema vinculante, basta seguir o mesmo padrão de algoritmo, descrito posteriormente nos tópicos Exemplo de Servidor em PHP e em Ruby (seção 6.5.1).

2.9 MANIPULAÇÃO DE IMAGEM E FILTROS

Uma imagem pode ser definida como uma função bidimensional, f(x, y), em que x e y são coordenadas espaciais (plano), e a amplitude de f em qualquer par de coordenadas (x, y) é chamada de intensidade ou nível de cinza da imagem nesse ponto. Quando x, y e os valores de intensidade de f são quantidades finitas e discretas, chamamos de imagem digital. O campo do processamento digital de imagens se refere ao processamento de imagens digitais por um computador digital. Observe que uma imagem digital é composta de um número finito de elementos, cada um com localização e valor específicos. Esses elementos são chamados de elementos pictóricos, elementos de imagem, pels e pixels. Pixel é o termo mais utilizado para representar os elementos de uma imagem digital (GONZALEZ E WOODS, 2010).

Segundo o Instituto de Nacional de Pesquisas Nacionais (INPE)[23], técnicas de filtragem e manipulação de imagens são transformações da imagem pixel a pixel, que não dependem apenas do nível de uma cor específica de um determinado pixel, mas também dos valores e níveis das cores dos pixels vizinhos.

[23] <http://www.dpi.inpe.br/spring/teoria/filtrage/filtragem.htm>. Acesso em 21/05/2017.

3 ESTADO DA ARTE

Neste capítulo serão abordados trabalhos correlatos, sendo apresentado os mais relevantes a essa pesquisa e desenvolvimento. Há muitas plataformas que se assemelham ao projeto referente, porém são poucas as que fazem a mesmas coisas com para as mesmas finalidades. O StudioJS, CamanJS e PIXLR foram referenciadas possuis juntas possibilitam abranger o máximo dos objetivos e características propostas pelo trabalho, sendo estes um minieditor, uma biblioteca e um estúdio de manipulação respectivamente.

3.1 O MINIEDITOR STUDIOJS

O primeiro trabalho de referência, se chama StudioJS (Figura 4), que trata-se de um projeto de desenvolvimento para vinculação ao HTML, com código aberto e gratuito do mesmo ramo e aplicação do trabalho proposto.

Possui ferramentas como: rotação; recorte; controle de brilho; controller de contraste; aplicação de filtros. Não possui ferramentas textuais como inserção e definição de estilos textuais (negrito, itálico, etc); também, pelo seu site[24] não foi possível efetuar a seleção de uma nova imagem para edição.

Essa ferramenta não utiliza nenhum *framework* popular para estilização da interface e segundo testes aplicados pelo autor (do trabalho referente) apresenta lentidão ao aplicar suas funcionalidades, levando até 5 segundos para efetuar uma única rotação da imagem[25].

Disponibilizado pela empresa Imazen, gerenciadora de *softwares* para processamento de imagem. Uma das principais vantagens do *framework*, é o fato dela estar apresentado no formato de um pequeno editor de imagens, possibilitando maior usabilidade ao usuário final.

---

[24] <http://studio.imageresizing.net/studio.html>. Acesso em 12/05/2017.

[25] Nos testes utilizou-se do navegador Google Chrome (versão 59) em um computador com processador Core i3 com 6 Gigas de memória RAM.

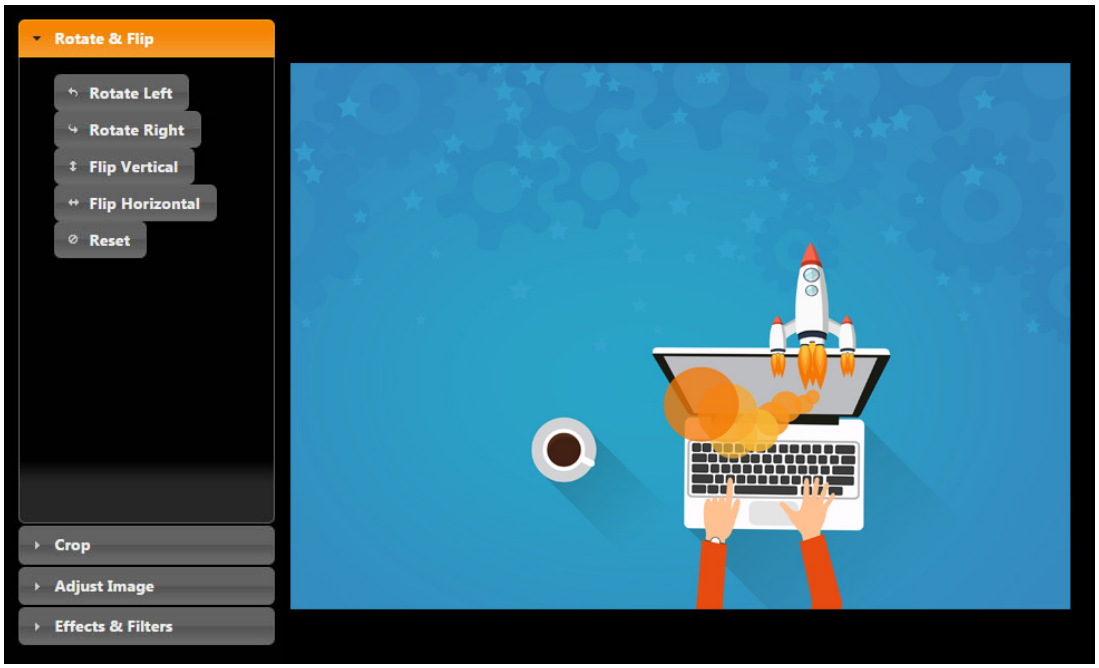

Figura 4 – Minieditor StudioJS

Fonte: Imazen, StudioJS[26].

3.2 A BIBLIOTECA CAMANJS

A biblioteca CamanJS foi desenvolvida por Ryan LeFevre. Este projeto conta com várias contribuições de outros programadores, assim como o trabalho de referência anterior (StudioJS).

Este recurso não está apresentado no formato de estúdio, mas sim como bibliotecas vinculáveis independentes. Ela oferece várias ferramentas de extrema eficiência perante edições básicas de imagens (que não manipulam os pixels vizinhos). Caso o desenvolvedor deseje compor um estúdio ou algo semelhante com estas ferramentas, deverá implementar sua própria solução.

Essa biblioteca foi implementada utilizando o recurso Canvas[27] do HTML 5, a mesma tecnologia que está proposta como solução para o projeto referente, porém o ImageJS

[26] <http://studio.imageresizing.net/studio.html>. Acesso em 22/05/2017.
[27] Segundo Marijn Haverbeke, Canvas é tela, um único elemento do Modelo de Objeto de Documentos (DOM) que encapsula uma imagem. Ele fornece uma interface de programação para desenhar formas para o espaço ocupado pelo nó.

possuirá uma gama maior de ferramentas e possibilidades de edição, pois o próprio usuário irá aplicar as ferramentas consecutivamente em cima da imagem importada. A Figura 5 representa uma tela com o funcionamento e algumas das opções da biblioteca CamanJS.

Figura 5 – Exemplo de Implementação da Biblioteca CamanJS

Fonte: CamanJS, Página Inicial.

3.3 O ESTÚDIO PIXLR

O estúdio PIXLR[28] trata-se de uma plataforma Web não vinculável, ou seja, é um recurso para edição de imagens *online*, onde o usuário envia suas imagens, as manipula, e recupera a imagem com as manipulações aplicadas. Desenvolvida pela empresa Autodesk[29], o estúdio PIXLR está disponível para utilização via aplicativo móvel, programa *desktop* ou sistema Web.

Sendo o seu *layout*[30] a principal referência para este projeto, o PIXLR possui uma interface de fácil compreensão e usabilidade pelo usuário final. Devido ao fato de ser uma

[28] <https://pixlr.com>. Acesso em 14/02/2017.
[29] <http://www.autodesk.com.br>. Acesso em 14/02/2017.
[30] *Layout*: Desenho, modelo da página Web.

ferramenta não *cross-browser*, sua biblioteca de opções é muita rica e disponibiliza várias opções para efetuar edição e manipulações nas imagens enviadas pelo usuário. Na Figura 6, é possível observar a tela de edição e algumas opções da ferramenta PIXLR.

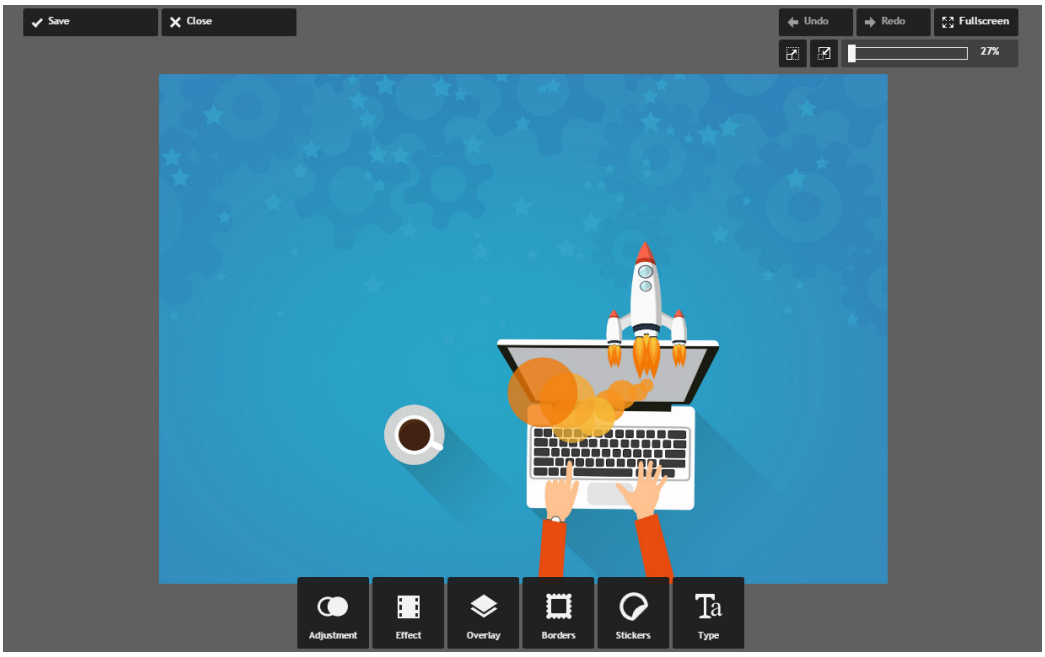

Figura 6 – Ferramenta PIXLR

Fonte: PIXLR, Web Apps[31].

3.4 COMPARATIVO ENTRE AS PLATAFORMAS

Na Tabela 2 pode-se verificar uma comparação sucinta das recursos disponibilizadas pelas plataformas, incluindo o trabalho referente, ImageJS – Baseando-se primordialmente nas características e recursos relevantes no trabalho proposto, tais atributos informados não representam a totalização das capacidades de cada ferramenta; portanto, não representa um comparativo de utilidade e sim de propriedades referentes ao ImageJS.

Previamente, verifique o detalhamento do elementos da tabela que estão marcados com asterisco (*):

[31] <https://pixlr.com/express/>Acesso em 22/05/2017.

- **Interface Responsiva:** No caso desse projeto, não é possível que o elemento canvas possui comportamentos responsivos, pois isso acarretaria na redistribuição de pixels do quadro caso a tela fosse redimensionada, isso resulta em perda das características lógicas da edição.
- **Atribuível em Formulário:** Permite que as edições concluídas sejam adicionadas a um formulário HTML.
- **Vinculável:** Possibilidade de incluir a ferramenta em uma plataforma Web externa, através da importação de arquivos na página do sistema vinculante.
- **Interface de Interação:** Consiste em possuir uma área visual de interação direta com o usuário, e não apenas uma biblioteca importável para auxiliar desenvolvedores, provendo recursos mediante a implementação.
- **Permite Colaboração:** Permitir que a comunidade de desenvolvedores interessada contribua com a plataforma através de ferramentas de desenvolvimento em grupo, como o Git ou o Subversion. Contexto referente ao tópico Contribua (seção 7.2).

Tabela 2 – Comparativo de Recursos Disponíveis entre as Plataformas

Recurso	STUDIOJS	CAMANJS	PIXLR	IMAGEJS
Brilho	Possui	Possui	Possui	Possui
Contraste	Possui	Possui	Possui	Possui
Borragem	Não Possui	Não Possui	Possui	Possui
Filtros	Possui	Não Possui	Possui	Possui
Cross-Browser	Possui	Possui	Não Possui	Possui
*** Interface Responsiva**	Não Possui	Não Possui	Não Possui	Possui
*** Atribuível em formulário**	Não Possui	Não Possui	Não Possui	Possui
*** Vinculável**	Possui	Possui	Não Possui	Possui
*** Interface de Interação**	Possui	Não Possui	Possui	Possui
*** Permite Colaboração**	Não Possui	Possui	Não Possui	Possui

4 ANÁLISE E PROJETO DE SISTEMA

Neste capítulo são descritos os procedimentos metodológicos da solução proposta para o problema apresentado. Os passos metodológicos estabelecidos inicialmente são relacionados e descritos na sequência.

4.1 LEVANTAMENTO DE REQUISITOS

Nesta etapa foram definidos os requisitos do sistema, ou seja, todas as opções e comportamentos que a plataforma deve possuir. Para isto foi efetuada uma pesquisa comparativa baseando-se em outros *softwares* da mesma área (assim como os citados no Capítulo 3, Estado da Arte) verificando quais são as ferramentas e interações presentes em comum nestas plataformas. Com isto, se obteve embasamento suficiente das possibilidades de implementação básicas nos editores de imagens, para assim desenvolver uma ferramenta de maior abrangência e genericidade.

4.1.1 DEFINIÇÃO DAS FERRAMENTAS

As ferramentas para edição de imagens são amplamente utilizadas para preparar imagens antes de se enviar em sistemas que são utilizados por vários usuários. Existem diversas ferramentas disponíveis para aplicar inúmeros tipos edições em imagens, desta maneira, para efetuar uma listagem das ferramentas mais relevantes para implementação, foram analisados alguns programas que já possuem popularidade neste ambiente de aplicação, tais como os citadas no capítulo Estado da Arte (seção 3); porém, para definir isso foi

necessário analisar sucintamente algumas plataformas a parte, como: Photoscape[32] e Instagram[33]. Na Tabela 3, são apresentadas as ferramentas para edição de imagens que são relevantes para implementação no editor.

Tabela 3 – Ferramentas Relevantes para Primeira Versão do Editor

EDIÇÃO	Recorte Live; Borragem Quadricular; Controle de Contraste; Controle de Brilho; Rotação para Esquerda.
FILTROS	O minieditor disponibiliza dois filtros desenvolvidos pelo autor da plataforma com objetivo sucinto de estudar e demonstrar técnicas em manipulação de imagens.

Notas:

(1) Vale ressaltar que a escrita deste documento de imediato não resultou na primeira versão da plataforma, apenas forneceu os principais recursos para se atingir tal objetivo. Sucedente, o tópico a seguir, assim como o tópico As CSS no Papel da JS (seção 5.3), oferecem uma melhor descrição sobre a complexibilidade de desenvolvimento da plataforma.

4.2 COMPLEXIBILIDADE DA WEB E DEFINIÇÃO DAS TECNOLOGIAS

Devido a complexibilidade, todo recurso desenvolvido para ambientes Web exige que o desenvolvedor esteja preparado para programar em mais de uma linguagem computacional; muitas vezes é necessário estudar outras áreas, como a segurança e versionamento de navegadores, ou configurações de servidores. Confira um trecho que melhor detalha essa complexibilidade tecnicamente:

> *Usuários da internet navegam em websites, que são escritos em basicamente 3 linguagens: (1) HTML, (2) CSS e (3) JavaScript. Estas 3 tecnologias são as linguagens básicas que todo navegador web ou browser precisa compreender. Programadores que desejam desenvolver sites precisam adquirir conhecimento*

[32] <http://www.photoscape.org/ps/main/index.php>. Acesso em 19/05/2017.
[33] <https://www.instagram.com/>. Acesso em 19/05/2017.

> *nestes linguagens. Todo código que se executa no browser é definido como camada cliente. Entretanto, o desenvolvimento de sistemas web dinâmicos – aqueles que alteram seu conteúdo de acordo com as ações do usuários – exige ainda o domínio de outras linguagens que são executadas no lado do servidor. São exemplos de linguagens da camada de servidor: PHP, Java, Ruby, Python, DotNet, etc.* (SCHEID, 2015)

Tendo em vista tais complicações, para que não se tenha imprevistos durante o desenvolvimento, é importante definir o máximo de tecnologias e escopos que serão estudados e utilizados no projeto em questão. As tecnologias utilizadas no trabalho referente foram listadas no capítulo Fundamentação Teórica (seção 2).

4.3 PROJETAR FILTROS

Os filtros que foram desenvolvidos no projeto são de implementação do autor; não focando apenas na aplicação de novas texturas, mas no desenvolvimento de algoritmos demonstrativos no processo de manipulação da imagem. Os filtros desenvolvidos objetivam a manipulação básica (ignora pixels vizinhos) das cores de cada pixel ou elemento da imagem. Dos 5 (cinco) filtros desenvolvidos, cada um deles salientou diferentemente a tonalidade RGB (vermelho, verde, azul; ou do inglês: *red*, *green, blue*). No tópico Filtros (capítulo de Desenvolvimento, seção 5.4.6), descreve melhor como foi desenvolvido cada filtro proposto pela plataforma.

4.4 PROJETAR INTERFACE

A interface é uma das principais etapas do desenvolvimento do projeto, pois é através dela que o usuário vai interagir com o sistema. Simplicidade e facilidade de uso são de extrema importância para que a ferramenta seja para o usuário uma etapa importante da interação com o sistema, e não apenas um estágio obrigatório, massivo e cansativo.

O desenvolvimento da interface será baseado nos estilos Material Design, desenvolvidos pela empresa Google, a qual já aplicou os princípios primordiais da IHC e continua a amadurecer a ferramenta junto à comunidade.

4.5 ÍCONE REPRESENTATIVO

É convencional toda programa ou produto possuir um ícone representativo, tanto por motivos de *marketing* quanto para fornecer uma imagem que facilmente recorda de tal software ou artefato.

O ícone do ImageJS foi construído se baseando em uma câmera no modelo de uma peça de um quebra-cabeça, que referencia as imagens que serão editadas e a característica vinculável do editor, que é capaz ser incorporado ou no caso “encaixado” em qualquer sistema Web; confira melhor o ícone representativo do ImageJS observando a Figura 7.

Figura 7 – ImageJS, Ícone Representativo

Fonte: Contribuição do Acadêmico Darlan Lara.

4.6 TELAS E MODELOS VISUAIS

Tendo em vista a usabilidade com a plataforma via dispositivos com telas menores (tablets e outros dispositivos móveis), planejou-se vários modelos visuais de interação até resultar no modelo adotado, tais como a interface de edição vertical referenciada no Apêndice A, Figura 27.

Devido a má usabilidade da plataforma utilizando a interface vertical em dispositivos com telas menores – em determinadas situações seria necessário criar uma barra de rolagem vertical para acessar as opções do menu, isso tomaria muito espaço horizontal da tela; portanto, optou-se por desenvolver uma nova interface distribuída horizontalmente, pode-se observar o resultado no Apêndice A, Figura 28.

Além do esquema visual estático do editor, houve também a necessidade de planejar e desenvolver o esquema de interações com a ferramenta, pois a tela do editor não é a única interface necessária para se utilizar de todos os recursos da plataforma. Há também outros elementos, tais como: Início de Edição e Edições Concluídas, Apresentação de Mensagens e Ferramentas Secundárias, que podem ser conferidos no Apêndice A, Figura 29 e 30 respectivamente.

4.7 PROJETAR RECURSOS LÓGICOS

Devido também à complexibilidade citada no tópico de Complexibilidade da Web [...] (seção 4.2) é importante direcionar de maneira sucinta um esforço prévio ao desenvolvimento, objetivando definições e planejamentos de como organizar os recursos programados (código escritos) de cada linguagem e tecnologia utilizada no projeto em questão, diminuindo assim a possibilidade de se deparar com bagunça excessiva de *scripts* ou até mesmo desentendimento de como deveria funcionar determinados blocos de códigos ou algoritmos.

Cada tecnologia de desenvolvimento possui padrões e métodos de organizar os códigos independentes e geralmente diferentes entre si – Devido ao escopo de cada linguagem, cada uma possui sintaxe distinta e com objetivos computacionais diferentes.

As principais tecnologias de desenvolvimento para o trabalho referente são o HTML, CSS e o JS, e foram organizadas com objetivo de proporcionar melhor funcionamento e clareza para uma plataforma Web dinâmica (que se altera mediante a interação do usuário). Como planejamento prévio ao desenvolvimento do ImageJS, algumas diretrizes foram definidas (salvas no caso de impossibilitação tecnológica):

- ➢ Efetuar o mínimo de interação com o documento ou DOM da página (alteração do HTML do documento origem);

- ➢ Todo conteúdo estático (HTML e CSS) deverá ser vinculado na página apenas uma vez, mantendo seu conteúdo durante as próximas solicitações de edição;

- ➢ Escrever toda a estrutura HTML e estilos CSS necessários para compor o estúdio diretamente na primeira requisição da plataforma (via solicitação do usuário);

- ➢ Todo conteúdo JavaScript somente deverá ser referenciado e processado pelo navegador via interação do usuário com a plataforma – blocos de códigos definidos em eventos lógicos do Agente Usuário podem ser afetados ou até mesmo acarretar na quebra do fluxo de eventos já definidos no sistema vinculante do estúdio.

5 DESENVOLVIMENTO

Neste capítulo será descrito todo projeto, dividindo e descrevendo cada fase bem como suas principais características em etapas. O desenvolvimento do sistema foi a etapa que mais concentrou os esforços dedicados ao trabalho, principalmente em pesquisas para encontrar determinados blocos de códigos.

É importante ressaltar que toda explicação voltada a conteúdo lógico de programação (códigos) será explicado de maneira genérica e não específica a cada instrução, pois seria irrelevante para o domínio deste documento, considerando a quantidade de códigos que o sistema possui. Para entender melhor o funcionamento lógico dos recursos do ImageJS, verifique no fim do documento o tópico Contribua.

5.1 HTML

O papel geral do HTML é de fornecer os recursos capazes de estruturar o corpo de todos os elementos do ImageJS, desde o estúdio de manipulação, botões, menus, até a interação com as edições já concluídas pelo usuário da ferramenta. Há algumas atuações específicas do HTML no projeto, tais como:

1. Durante as edições que serão efetuadas, fornecer o elemento Canvas do HTML5, servindo como área de escrita e manipulação da imagem selecionada;
2. Fornecer o componente de entrada de dados (*Input* do tipo *file,* arquivo) que receberá a imagem enviada do dispositivo do usuário;
3. Viabilizar o retorno textual das edições concluídas pelo usuário, encriptado no formato Base64 e preenchido em entradas de dados do tipo textual (*Input* do tipo *text*, texto).

5.1.1 AUTONOMIA LÓGICA

Há a possibilidade que uma ou mais *tags* HTML utilizadas no ImageJS já se encontre logicamente escrita e editada pelo sistema vinculante do estúdio – por exemplo a *tag* Div que é o elemento base fornecido pelo HTML; isso acarretaria no mal funcionamento da estrutura do estúdio, pois não há como prever se as programações do sistema vinculante seriam compatíveis com a estrutura do editor.

Todas as tecnologias do ImageJS devem possuir essa "Autonomia Lógica", pois seus comportamentos não devem influenciar diretamente nos recursos do sistema vinculante, o contrário também é válido. Porém, a importância da autonomia no âmbito estrutural dos elementos (HTML) será o grande impactante na autonomia geral do estúdio, pois as CSS e JS baseiam seus comportamentos nesta estrutura.

O desafio de tornar a estrutura do ImageJS totalmente independente do sistema que vinculará o estúdio se encontra na maneira que está nomeado os recursos HTML (elementos e atributos); portanto, para a escrita lógica de tais recursos, foram utilizados de *tags* e atributos específicos muitas vezes prefixadas por "imagejs" e criados especificamente para utilização da ferramenta (ou seja, tais *tags* não fazem parte do escopo original do HTML), tais como a *tag* ImageJS (escrita em uma página HTML como "<image-js>"). O detalhamento descritivo de como ficou a solução está escrito no tópico de Estrutura dos Elementos e Nomenclaturas (seção 5.1.2). Vale ressaltar que toda a estritura do editor abre junto com a página do sistema vinculante, semelhante a estrutura de um *modal*.

5.1.2 ESTRUTURA DOS ELEMENTOS E NOMENCLATURAS

Como previamente informado no tópico de Autonomia Lógica, todos os elementos do ImageJS que fazem referência a estruturamento HTML, foram escritos objetivando a independência lógica dos recursos perante o sistema vinculante. Neste tópico será descrito sucintamente a função de cada *tag* do ImageJS. Para verificar como foi ficou o resultado das estruturas em código HTML, observe o Apêndice B, Figura 31 e 32**.**

O ImageJS está dividido em dois conjuntos de elementos: os de vinculação e os de edição, que são representados respectivamente pelas duas principais *tags* da plataforma: a *tag* ImageJS (“<image-js>” em HTML), e a *tag* Studio (“<studio>” em HTML). Todas as outras *tags* são para estruturar as ferramentas e os diversos elementos de interação com o estúdio. Na Tabela 4, estão listados todas as *tags* HTML desenvolvidas até o momento para compor o editor. Para ter acesso a toda a estrutura CSS do projeto, efetue leitura sucedente no tópico Contribua.

Tabela 4 – Tags HTML do ImageJS

II	TAG	DESCRIÇÃO
1	<image-js>	*Tag* de responsável pela incorporação do estúdio, basta escrevê-la sem conteúdo algum no DOM da página que deseja vincular para que o estúdio seja incorporado (como demonstrado anteriormente pela Figura 1). Após a inicialização automática e durante as interações com a plataforma de edição, esse elemento será preenchido com elementos de inicialização de edição e edições concluídas.
1.1	<editions>	Recipiente de edições concluídas pela usuário.
1.1.1	<edition>	Elemento que representa uma edição concluída pela usuário utilizando a plataforma.
1.1.2	<edition name=”new”>	Elemento de edição com objetivo de iniciar o estúdio, possui a mesma *tag* por estar no mesmo escopo de elementos.
2	<studio>	*Tag* recipiente de todos os recursos de edição, desde o estúdio interativo até as ferramentas. O conteúdo desse elemento será vinculado a pág. HTML quando o usuário requisitar a primeira edição, após isso apenas será manipulado.
2.1	<tools-primary>	Representa o recipiente de ferramentas, mensagens e interações visuais Primárias, como envio de endereços de imagens da Web.
2.1.1	<tools>	Representa um grupo de ferramentas especificadas pelo atributo “name”.
2.1.1.1	<tool>	Representa uma ferramenta do sistema, seja de edição ou informativa.

2.1.2	<info>	Representa uma mensagem informativa do editor.
2.1.2.1	<centralizer>	*Tag* responsável por centralizar totalmente (em sentido vertical e horizontal, sem perda de proporções) um outro elemento em função do seu elemento recipiente.
2.2	<tools-secondary>	Representa o recipiente de ferramentas, mensagens e interações visuais Secundárias, como subopções de ferramentas primárias.
2.3	<editing>	Recipiente dos elementos responsáveis pela edição atual, tais como: a imagem de edição; o Canvas de manipulação; seletores e utilitários internos.
2.3.1	<container>	Saguão de edição, do canvas e dos seus seletores de áreas. Esse elemento limita os outros que interagem com o canvas a não saiam do espaço em tela dedicado a edição – impossibilitante por exemplo que ao utilizar a ferramenta de recorte seu mouse saia até o menu primário.
2.3.1.1	<selector>	Seletor translúcido do canvas. Utilizado por exemplo pela ferramenta de recorte, para especificar no formato de retângulo qual a área que se deseja o obter.

Notas:

(1) II ou Índice de Indentação faz referência aos códigos presentes nos apêndices do contexto em questão.

5.2 CSS

Neste projeto a CSS será responsável por dar estilo a todo conteúdo estrutural do ImageJS, desde cores, bordas, aspecto dos botões, menus, posicionamento, comportamento de alguns elementos HTML, entre outros aspectos visuais. A maneira que está estruturado a CSS do projeto define o conforto visual e o nível de usabilidade que o usuário terá durante a utilização do estúdio, proporcionando uma experiência satisfatória.

5.2.1 AUTONOMIA LÓGICA

Assim como já descrito no tópico de Autonomia Lógica do estrutural (do HTML), há necessidade que os códigos de estilização (as CSS) impliquem diretamente o sistema vinculante, aplicando estilos a elementos externos aos do ImageJS. Para isso os estilos devem ser escritos de maneira centralizada a estrutura do ImageJS – criação de classes CSS e estilos baseados em atributos correm este risco; portanto, a solução de estilos baseado diretamente nas *tags* da plataforma foi aderida. O tópico de Estilização Centralizada em *Tags* (seção 5.2.1.1) descreve tal solução.

5.2.1.1 ESTILIZAÇÃO CENTRALIZADA EM TAGS

A estilização centralizada em *tags* consiste em desenvolver os recursos CSS sem a utilização exagerada dos recursos que a tecnologia provém (seletores por ID, Classes ou Atributos), quanto menos explorar tais recursos, maior Autonomia Lógica do editor em função do sistema vinculante, pois podem existir IDs, Classes e Atributos nomeados da mesma maneira entre as plataformas. Portanto, todos os estilos do ImageJS foram aplicados diretamente em suas tags HTML. Pode-se observar um trecho de código CSS referente ao projeto no Apêndice C, Figura 34. Para ter acesso a toda a estrutura CSS do projeto, efetue leitura sucedente no tópico Contribua.

5.2.2 CENTRALIZAÇÃO CROSS-BROWSER

No escopo visual da plataforma ImageJS, a centralização total (em sentido vertical e horizontal) dos elementos em função dos seus recipientes tornou-se uma necessidade estética, pois deixar todos os componentes visuais alinhados nos cantos da tela em muitos casos não transpareceu uma boa organização visual.

Há várias maneiras de se resolver tal problema, porém alguns não demonstram muita eficiência perante a compatibilidade multi-navegador, e outros funcionam apenas para casos à parte. As maneiras as testadas no desenvolvimento do projeto para resolver o problema de centralização foram:

- Utilizar o atributo “position” preenchido com o valor “absolute” (disponível desde as primeiras versões do CSS segundo a API da Mozilla Foundation[34]) com os valores de margens preenchidos com valores fixos; isso funcionaria apenas em telas com o tamanho programado inicialmente. Observe um exemplo na Figura 8.

```
centralizer {
    position: absolute;
    top: 10px;
    bottom: 10px;
    right: 10px;
    left: 10px;
}
```

Figura 8 – Exemplo de Posicionamento Estático

Fonte: Autor.

- Outra solução que possível e que não apresenta problemas com tamanhos diversos de telas seria utilizando o posicionamento baseado em porcentagem. Observe um exemplo na Figura 9;

```
centralizer {
    margin-top: 50%;
    margin-bottom: 50%;
    margin-left: 50%;
    margin-right: 50%;
}
```

Figura 9 – Exemplo de Posicionamento por Porcentagem

Fonte: Autor.

[34] <https://developer.mozilla.org/pt-BR/docs/Web>. Acesso 25/05/2017.

A solução de centralização de elementos baseada em porcentagem seria o suficiente para o projeto referente se não tivesse apresentado comportamento divergente com o navegador Mozilla Firefox (na versão 52 de 32 *bits*); tendo em vista esse diferencial lógico do atributo e a importância do Firefox para o escopo de navegadores, adotou-se outro método de centralização buscando o funcionamento nos navegadores de grande popularidade atualmente (seção 6.1.1). Confira a solução de centralização adotada no tópico O Atributo Transform (seção 5.2.2.1).

5.2.2.1 O ATRIBUTO TRANSFORM

Em testes efetuados pelo autor, no escopo do ImageJS, a implementação através do atributo “transform” disponível na versão 3 do CSS, apresentou melhor compatibilidade entre os principais navegadores Web (referenciados no tópico de Centralização Cross-Browser, seção 5.2.2).

Segundo a API da Mozilla, a propriedade CSS “transform” permite modificar o espaço coordenado do modelo de formatação CSS. Usando-a, elementos podem ser traduzidos, rotacionados, ter seu tamanho ajustado e inclinados de acordo com os valores definidos. Observe no Apêndice C a Figura 35 e 36, demonstrando respectivamente a solução implementada na plataforma referente e o resultado visual.

5.3 AS CSS NO PAPEL DA JS

A complexibilidade de se desenvolver um estúdio para edição de imagens desde seu estágio inicial, não se encontra na implementação de algoritmos para ferramentas de edição, mas nas características mais triviais da plataforma, tais como a importação da imagem para o ambiente interno do editor.

A escrita da imagem no canvas é uma responsabilidade dinâmica do JavaScript; porém durante esse processo há muitas informações que devem ser previamente preenchidas –

trata-se de um elemento estrutural (estático) do HTML, seus dados lógicos são informados durante o carregamento da página no navegador, sem conhecimento das futuras imagens editadas.

De forma simples e pouco proveitosa dos recursos disponíveis, utiliza-se do JavaScript para calcular parâmetros como o tamanho da tela do cliente e da imagem a ser vinculada (sempre relacionando as proporções verticais e horizontais), para assim atribuir a imagem corretamente no Canvas e no escopo lógico (valores e atributos da imagem) do Editor.

Por se tratar de funcionalidades básicas e primordiais (como a vinculação da imagem) há o interesse de se obter soluções simples, eficazes e ao mesmo tempo criativas (quando possível); portanto, optou-se por pesquisar maneiras divergentes da simples porém trabalhosa programação com JS, neste momento que as CSS tiveram grande impacto positivo no desenvolvimento do sistema, trabalhando em conjunto com o JS.

De imediato haviam dois grandes problemas com a obtenção e escrita de uma determinada imagem no quadro de edições (Canvas): proporção e resolução (ou qualidade), pode-se conferir com melhor detalhamento nas seções 5.3.1 e 5.3.2 respectivamente.

5.3.1 PROPORÇÃO DA IMAGEM NO CANVAS

Manter a proporção (relação de distribuição vertical e horizontal) da imagem não trata-se apenas de um problema estético, mas lógico, pois cada pixel do quadro de edições representa um valor interno a ser manipulado; ou seja, proporção inválida resulta numa imagem totalmente diferente observando seus valores computacionais.

Como manter as proporções da imagem (relação de distribuição vertical e horizontal), maximizando o máximo possível em função da tela do dispositivo de acesso, seja uma tela maior verticalmente ou horizontalmente. Observe na Figura 10 o exemplo de imagem proporcionada e desproporcionada.

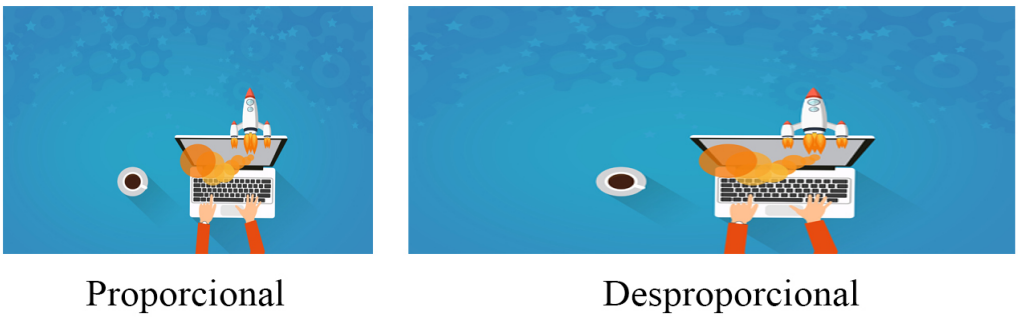

Figura 10 – Exemplo de Imagem Proporcional e Desproporcional

Fonte: Autor.

Solução: Para resolver este problema de proporções, utilizou-se do atributo CSS "object-fit" (referente ao comportamento e posicionamento de uma determinada imagem em função do seu recipiente) integrado com o valor "cover" ou tampa – segundo a API da Mozilla[35], o atributo consiste no ajuste de objeto e especifica como o conteúdo de um elemento substituído deve ser ajustado à caixa estabelecida pela altura e largura utilizadas; o valor "cover" representa que conteúdo substituído é dimensionado para manter sua relação de aspecto ao preencher toda a caixa de conteúdo do elemento: seu tamanho de objeto concreto é resolvido como uma restrição de cobertura perante a largura e altura utilizadas pelo elemento (não causa percas nem recortes na imagem). Em resumo, a utilização desses atributo faz com que a imagem preencha o seu elemento recipiente o máximo possível sem perder suas proporções verticais e horizontais. Observe na Figura 11 como ficou a implementação dessa solução na plataforma em questão.

```
studio editing img {
    object-fit: cover !important;
    max-width: 90% !important;
    max-height: 90% !important;
}
```

Figura 11 – Solução Prévia, O CSS de Proporção

Fonte: Autor.

[35] <https://developer.mozilla.org/pt-BR/docs/Web/CSS/object-fit>. Acesso em 26/05/2017.

Mas para resolver o esquema as proporções não foi o suficiente apenas ter conhecimento desse poderoso recurso do CSS – pois o mesmo é aplicado em imagens e o elemento Canvas não é uma imagem, e sim um quadro de manipulação.

Utilizou-se em conjunto com o Canvas o a tag “img” (<img>) do HTML, com índice de indentação 2.0.7 representado pela Figura 32 no Apêndice B. Dessa maneira, quando a imagem é enviada ao estúdio, primeiramente a JavaScript a escreve neste elemento (por padrão oculto), logo as CSS preenchem as propriedades com o atributos “object-fit”, em seguida a JS obtém os dados do elemento “img” e os escreve no Canvas, possibilitando assim que a imagem seja escrita no canvas com as proporções originais.

5.3.2 QUALIDADE DA IMAGEM NO CANVAS

Ao carregar a plataforma o Canvas de edição não tem os valores das imagens que serão editadas pois ainda não foram enviadas; portanto, ele inicia e preenche suas propriedades coma a Altura (*height*) e Largura (*width*) proporcionalmente diferente da imagem que será enviada, geralmente zerados ou com valores proporcionais inferiores; logo, a imagem será escrita numa “tela” ou “espaço” (canvas) menor que sua origem, causando perda da resolução ou qualidade. Na Figura 12, pode-se observar o efeito da perda de qualidade da imagem importada em um Canvas com essas circunstâncias.

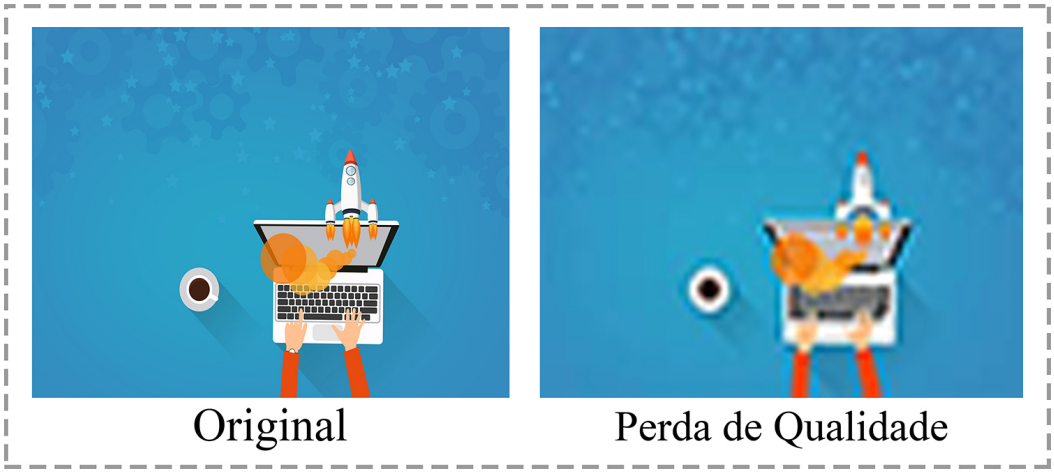

Figura 12 – Perda de Resolução

Fonte: Autor.

Para solucionar tal problema de resolução não houve necessidade de pesquisar a fundo outras possibilidades, bastou ter uma visão básica dos recursos ao redor. Se a imagem perde a resolução devido ao fato do canvas iniciar vazio, basta então não deixar o elemento iniciar vazio; portanto, as propriedades de estilização *height* e *width* do Canvas receberam um valor padrão de inicialização, 5000 (cinco mil pixels).

Qualquer imagem enviada com proporções vertical e horizontal inferior a 5000px não acarretará na perda de qualidade. Esse valor é relevante devido a exigência computacional que seria de manipular uma imagem maior que 5000px proporcionais; dessa maneira, qualquer imagem enviada acima da proporção definida perderá parte da sua resolução (isto pode ser editado conforme os objetivos de vinculação do estúdio). Para verificar essa solução em modelo estrutural, configura o elemento com Índice de Indentação 2.0.6 na Figura 32, Apêndice B.

5.4 JAVASCRIPT

No projeto referente, a linguagem JavaScript terá como responsabilidade todo o escopo dinâmico do *framework* ImageJS, tais como: a manipulação do HTML do estúdio; a vinculação da imagem no editor; aplicação de filtros das ferramentas de edição; entre outras funcionalidades dinâmicas.

5.4.1 AUTONOMIA LÓGICA

Como já descrito nos tópicos de Autonomia Lógica referente ao HTML e CSS (seções 5.1.1 e 5.2.1 respectivamente), há também a necessidade que os recursos do JavaScript trabalhem de maneira independente do sistema vinculante. Tendo em vista que o HTML e CSS já estão implementados visando tal característica, resta apenas agora que o JS utilize corretamente os elementos estruturais. Para isso foi adotado um métodos de desenvolvimento

externo aos eventos do navegador (descrito no tópico de Eventos, seção 5.4.2) e em escopo estrutural centralizou-se os recursos lógicos em um único objeto com escopo isolado (descrito no tópico Modelo Construtivo, seção 5.4.3).

5.4.2 EVENTOS

Tendo em vista que o JavaScript atribuído ao navegador nos permite manipular eventos globais da página em questão (tais como “onload”, “onchange”), estes recursos do ImageJS foram desenvolvidos divergindo-se de soluções via eventos do navegador, pois há um risco muito grande de que o sistema vinculante já possua valores lógicos nestes eventos, sendo afetados então pelos possíveis *scripts* do editor. Todos os eventos do ImageJS são internos ao seu escopo, eventuando apenas a própria estrutura de elementos, não há componentes globais no escopo lógico do estúdio, além da sua variável de instância.

O ImageJS é instanciado junto com o DOM da página, não necessariamente interno a um evento de leitura, mas no momento exato em que o código é atribuído ao escopo do sistema vinculante. Todo seu recurso lógico fica disponível através de uma variável de instância global chamada “imageJS”, disponibilizando valores e métodos via chamadas de acesso construtivo, detalhado na to tópico Modelo Construtivo (seção 5.4.3).

5.4.3 MODELO CONSTRUTIVO

Definir a organização lógica dos recursos programados é um estágio bastante importante considerando como será o resultado final do código. Em JavaScript há definitivamente várias maneiras de se organizar logicamente os recursos, as relevantes para o projeto em tratado são:

1. Escrevendo todo o código de maneira estruturada;
2. Criando funções globais e posteriormente apenas referenciá-las;
3. Criar funções e atribuir em variáveis globais ou não;

4. Criar um objeto fábrica (ou *factory*), utilizando *hashes*;
5. Criar um objeto utilizando o modelo Construtivo, instanciando um objeto a partir de uma função.

Tendo em vista os princípios da POO (Programação Orientada a Objetos), na plataforma em questão, o quinto método teve melhor *design* lógico, pois permite a instanciação de um objeto JS onde o escopo interno dos métodos e atributos são escritos utilizando a variável local “this”, nativa no escopo interno de cada função do JS. Para entender como resultou-se o esquema lógica estrutural de escrita, instanciação e chamada dos métodos da plataforma, observe a Figura 37, presente no Apêndice D.

5.4.4 ESTRUTURA DO OBJETO

A estrutura lógica do objeto consiste em descrever como seu conteúdo está organizado, narrando os métodos, variáveis e configurações da instância da plataforma.

Basicamente, o ImageJS possui várias propriedades lógicas, desde variáveis constantes a métodos ricos em instruções.

Métodos: Os métodos do objeto nomeado como “imageJS”, são divididos entre utilitários e ferramentas (*utils* e *tools* respectivamente); utilitários são métodos que tem como objeto recortar algum valor, manipular um elemento ou resumir trecho de outro método, são métodos de auxílio; os métodos ferramentas, representam blocos instrucionais que são direcionados a manipulação do quadro de edição, aplicando filtros, propriedades, entre outras funções.

Variáveis: A maioria das variáveis da instância tem como objetivo armazenar referências para objetos estruturais (elementos HTML) do ambiente da plataforma. Porém, salvam também dados de edição tais como referências lógicas para o Canvas e seu contexto, ou até mesmo contadores de edição. No ambiente interno do objeto há variáveis prefixadas por cifrão ($variavel), que representam a referência

estrutural. As demais são auxiliares para as instruções de edição e interação com o editor.

Variáveis de Ambiente: Com objetivo de descrever uma configuração ou característica impactante em determinadas instruções do estúdio (que podem variar de acordo necessidade e intenções levadas a importação no sistema vinculante), as variáveis de ambiente são constantes e globais ao escopo da página. Até o momento, o editor possui apenas alocação com estas característica, que faz referência à definição do ambiente de produção ou desenvolvimento da plataforma (leia mais no tópico Ambiente de Produção e Desenvolvimento, seção 5.5); nomeada como IMAGEJS_ENVIROMENT seus dois possíveis valores são textuais, “production” e “development”, representando o ambiente de produção ou desenvolvimento, respectivamente.

Na Tabela 5, pode-se observar uma lista sucintamente descrita de quais método e variáveis compõem a instância da plataforma até o momento. Para entender melhor cada estrutura ou variável de lista, efetua a leitura sucedente do tópico Contribua.

Tabela 5 – Lista de Variáveis e Métodos da Classe ImageJS

VARIÁVEL	MÉTODO	F/R[36]	DESCRIÇÃO
IMAGEJS_ENVIROMENT		string	Constante que define o ambiente,
html		object	Objeto com o conteúdo HTML armazenado por índice.
css		string	Conteúdo CSS da plataforma.
tags		object	Objeto com as principais tags armazenada por índice.
count		int	Totalizador de edições referente a sessão de acesso ao navegador.
origin		string	Endereço local ou Web da imagem que está sendo editada.
$editions		jQuery	Referência para o elemento estrutural <editions>.

[36] F/R significa Formato/Retorno, referenciando a variável ou método em questão.

$studio		jQuery	Referência para o elemento estrutural <studio>.
$container		jQuery	Referência para o elemento estrutural <container>.
$canvas		jQuery	Referência para o elemento estrutural <canvas>.
$image		jQuery	Referência para o elemento estrutural <img>.
canvas		Canvas	Referência do objeto Canvas.
context		Context2D	Referência do objeto Context2D do canvas em questão..
	writeDOMComponent()	void	Escreve todos os componentes no sistema vinculante e armazena suas referências.
	edit(image_path)	void	Inicia a edição de uma determinada imagem.
	open()	void	Abre o editor.
	close()	void	Fecha o editor.
	hasEdition()	void	Verifica se há alguma imagem sendo editada no estúdio.
	rand(size)	string	Gera uma string “par” aleatória.
	message(name)	void	Apresenta uma mensagem no editor.
	tools(name)	void	Apresenta um grupo de ferramentas no editor.
	data()	object	Obtém ou Escreve dados no Canvas.
	write(function(pixels))	void	Cria e aplica um algoritmo e edição.
	download()	void	Faz *download* da edição atual.
	cancel()	void	Cancela a edição.
	conclude()	void	Conclui a edição.
	upDevise()	void	Solicita os elementos para enviar uma imagem do dispositivo.
	upWeb()	void	Solicita os elementos para enviar uma imagem da Web.
	resizable()	void	Permite que o Canvas se redimensione de acordo com o tamanho da janela do navegador.
	fullScreen()	void	Coloca e retira o editor de tela cheia.
	reset()	void	Reseta a imagem editada para seu estado origem.
	crop()	void	Recorta a imagem.
	brightness()	void	Aplica o efeito de brilho na imagem.
	contrast()	void	Aplica o efeito de contraste na imagem.

	blur()	void	Aplica o efeito de borragem na imagem.
	filters()	object	Aplica várias filtros personalizados na imagem, separados por índice do objeto.

Notas:

(1) Para acessar qualquer valor (método ou variável) da instância da classe ImageJS, basta fazer referência lógica através da variável “imageJS”, por exemplo: imageJS.método().

5.4.5 A ESCRITA DA IMAGEM

A escrita da imagem com toda certeza é o passo com maior importância no escopo lógico do ImageJS, trata-se do momento que a imagem original é escrita no quadro de edição, a ação primordial que permite que todas as ferramentas trabalhem no quadro.

Inserir a imagem no Canvas também não foi uma solução simples, havia a dificuldades de a escrita completada da imagem sem recortar seus extremos, e é nesse momento que o JS irá utilizar dos recursos que a CSS do projeto provém (como descrito no tópico Proporção da Imagem no Canvas, seção 5.3.1) – primeiramente escrevendo a imagem no elemento “<img>” oculto e após isso copiar suas propriedades para o Canvas.

O método utilizado para efetuar a escrita chama-se “drawImage“ e é nativo da API JavaScript, pertence ao objeto “context” de um determinado Canvas (“*context*” ou contexto, representa o escopo de um quadro em questão, possibilitando que se trabalhe com vários Canvas em um único escopo lógico) . Baseando-se na documentação *online* da Mozilla[37], o método possui em sua totalidade nove parâmetros. Para resolver o problema de escrita no ImageJS todos os nove foram utilizados, obtendo dados do elemento oculto “<img>” (preenchida anteriormente com CSS) e propriedades da imagem em questão. A Figura 38, no Apêndice D apresenta o código escrito para implementar a solução.

[37] <https://developer.mozilla.org/pt-BR/docs/Web/API/CanvasRenderingContext2D/drawImage>. Acesso em 28/05/2017.

5.4.6 FILTROS

Como descrito previamente no tópico Projetar Filtros (seção 4.3), a plataforma propõe o desenvolvimento de filtros que manipulem os pixels do canvas de maneira a dar outro aspecto a aos dados visuais da imagem que sendo editada.

Na primeira versão deste documento foi proposto o desenvolvimento de ao menos dois filtros com algoritmos proprietários para manipulação do canvas, porém, até então já foram desenvolvidos cinco métodos, objetivando o básico da manipulação.

Os filtros aplicam efeitos como o aumento do tom vermelho, verde e azul da imagem, outro filtro personaliza os três tons RGB destacando em tom de roxo; por fim, o filtro que aplica o efeito “cinza” ou preto e branco. Na Figura 13 pode verificar visualmente com é o resultado dos 5 filtros aplicados em um única imagem.

Figura 13 – Os Cinco Filtros Desenvolvidos (Vermelho, Verde, Azul, Roxo, Cinza)

Fonte: Autor.

Na Tabela 6, pode-se ler a descrição sucinta do algoritmo de cada filtro desenvolvido; para melhor entendimento e visualização do código JS puro dos filtros, observe o tópico sucedente Contribua.

Tabela 6 – Descrição Lógica dos 5 Filtros Implementados[38]

NOME	PRÉVIA DO ALGORITMO	DESCRIÇÃO
red	pixels[i] += 25;	Basta salientar o índice RGB referente a cor vermelha do pixel em questão.
green	pixels[i + 1] += 25;	Basta salientar o índice RGB referente a cor verde do pixel em questão.
blue	pixels[i + 2] += 25;	Basta salientar o índice RGB referente a cor azul do pixel em questão.

custom/purple	pixels[i] += 10;
	pixels[i + 1] += -20;
	pixels[i + 2] += 10;
		Basta salientar o índice RGB referente as cores vermelha e azul ao mesmo tempo em que diminui o tom verde.

grey/blackAndWhite	var grey = parseInt((pixels[i] + pixels[i + 1] + pixels[i + 2]) / 3);	Para isso é necessário obter a média de todos os toms RGB do pixel em questão, mediante a isso escreve o mesmo valor para os três índices RGB.

Fonte: Autor.

5.4.7 SEGURANÇA DO NAVEGADOR

Além dos nossos dados pessoais, acessamos também através dos navegadores diversos dados privados de outras páginas. Devido a essa larga escala de dados não proprietários que circulam pelo seu domínio, os navegadores desenvolveram vários princípios visando a segurança de tais informações. Cada um possui os próprios preceitos de segurança, porém, há outras regras que são globais, pois tratam-se definições da tecnologia e esquemas básicos de segurança.

Os princípios definidos na API online da Mozilla referente ao Controle de Acesso, e algumas especificações do Elemento Input do HTML, foram as características de segurança que implicaram no projeto em questão, pois restringem a escrita de imagem a partir da Web e

[38] Para entender o conteúdo completo dos algoritmos referenciados, basta efetuar leitura sucedente no tópico referente de criar edições com a plataforma (seção 6.6).

a inserção de edições concluídas no formulário do sistema vinculante. Nos tópicos seguir há uma melhor descrição.

5.4.7.1 COMPARTILHAMENTO DE RECURSOS ENTRE ORIGENS (CORS)

De princípio, o minieditor ImageJS pretendia funcionar acessando seu sistema vinculante utilizando não apenas o protocolo HTTP, mas o FILE também (que consiste apenas em abrir um arquivo no navegador, sem necessidade de requisição a um servidor). Porém, o objeto XMLHttpRequest utilizado pela plataforma durante o processo de vinculação de imagens da Web, possui algumas restrições de requisição quanto trata-se de um domínio solicitando dados de outro – Quando utilizado o protocolo File, o navegador não fornece informações de origem ao objeto de requisição, pois não há servidor interpretando a página, portanto, o domínio requisitado não permite a transferência do arquivo pois não sabem qual é a origem da requisição.

Inicialmente, mesmo com a utilização de servidores não era permitido a requisição de arquivos entre origens diferentes. Como descreve a API online da Mozilla[39]: Um recurso realiza uma requisição HTTP entre origens quando este solicita um recurso de um domínio diferente da origem. Por exemplo, uma página HTML servida pelo dominio http://domain-a.com faz uma solicitação <img> src para o domínio http://domain-b.com/image.jpg. Por razões de segurança, os navegadores restringem solicitações HTTP de origem cruzada iniciadas a partir de *scripts*. Por exemplo, *XMLHttpRequest* e Fetch seguem a política de mesma origem. Porém, após a solicitação de desenvolvedores os navegadores Web, o Mecanismo de Compartilhamento de Recursos entre Origens (CORS) foi desenvolvido, este mecanismo fornece aos servidores de web controles de acesso entre domínios, que permitem transferências seguras de dados entre domínios. Os navegadores modernos usam o CORS em um container API – como XMLHttpRequest ou Fetch - Para mitigar os riscos de pedidos HTTP de origem cruzada.

Com isso, a requisição de imagem para edição através de URL externa é possível, porém apenas em um ambiente com as informações de origem providas corretamente ao

[39] <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Controle_Acesso_CORS>. Acesso em 05/06/2017.

objeto de requisição (XMLHttpRequest); ou seja, todo recurso com requisições cruzadas somente funcionará se o sistema vinculante estiver sendo interpretado por um servidor HTTP.

5.4.7.2 HTML INPUT ELEMENT

Quando o usuário conclui a edição de uma determinada imagem na plataforma, o editor gera um arquivo lógico (Blob, seção 5.4.8) referente a esta edição, este arquivo deveria ser escrito posteriormente em um Input HTML do tipo *File*, para então o usuário enviar o formulário ao servidor com todas as edições concluídas; porém, até o momento, por questões de segurança, isso não é possível.

Na página da API online da Mozilla[40], que destaca as propriedades do elemento Input, é possível observar o que a propriedade *file* (referente a um ou mais arquivo vinculados ao elemento) está marcada como "*readonly*" (somente leitura), isso significa que não é possível escrever valores nesta propriedade, pode-se apenas obter um cópia para leitura e manipulação dos dados.

Após pesquisas realizadas pelo autor, não foi encontrada a origem explicativa referente a característica de somente leitura para a propriedade de arquivos. Porém, pode-se considerar um motivo importante: se a escrita de arquivos em elementos Inputs fosse permitida, seria simples se obter dados e informações importantes do computador do usuário, já que bastava apenas escrevê-los no formulário da página.

Com isso, para resolver este problema o ImageJS implementou uma solução própria porém não tão simples como escrever o arquivo no formulário. Para atribuir as edições ao formulário, primeiramente a plataforma converte a edição para texto puro utilizando o formato de codificação de dados Base64, que permite efetuar essa conversão sem que se perca as características (propriedades e valores) do arquivo em questão. O tópico seguinte, Envio de Múltiplas Edições (seção 5.4.8), descreve essa solução adotada pelo minieditor.

[40] <https://developer.mozilla.org/pt-BR/docs/Web/API/HTMLInputElement>. Acesso em 05/06/2017.

5.4.8 O ENVIO DE MÚLTIPLAS EDIÇÕES

A pós o usuário terminar as edições na imagem, além da possibilidade de *download* da imagem editada, a plataforma permite que o usuário conclua a edição e vincule-a no formulário da sua página, e caso necessário, efetue novas edições em outras imagens. Para isso é necessário que após a conclusão da edição, o arquivo correspondente a imagem editada não fosse submetido a um endereço Web de destino diretamente, como deveria ser feito caso a edição resultasse em um arquivo no formato de imagem (como PNG ou JPEG).

Para possibilitar que o usuário efetue quantas edições forem necessárias, o editor converte então o resultado do Canvas para um arquivo lógico no formato Blob – Segundo a API online da Mozilla, o método toBlob()[41] do Canvas, cria um objeto representando a imagem contida no quadro de edição; este arquivo pode ser armazenado em cache no disco ou na memória, a critério do agente do usuário.

Para cada edição concluída, o minieditor obtém o objeto Blob e converte seus valores para texto Base64. Este texto será escrito em um elemento do Input (respectivo a edição em questão) internamente ao formulário vinculado, permitindo assim que o usuário envie o formulário com o conteúdo textual de todas as edições. O fato das edições serem enviadas no formato textual, exigirá que o servidor que receberá os dados enviado pelo formulário converta as propriedades textuais Base64 para o arquivo representado pelo valor em questão. Os dados codificados nesse formato provém as informações necessárias para conversão. Pode-se observar exemplos no tópico Escopo do Servidor (seção 6.5).

5.5 AMBIENTE DE PRODUÇÃO E DESENVOLVIMENTO

Este projeto é desenvolvido focando a evolução contínua das suas bibliotecas, sua estrutura lógica permite que sempre se esteja criando novas ferramentas e opções; o enriquecimento de uma aplicação jamais deve parar, as tecnologias estão evoluindo, e nisso o projeto também deve crescer.

---

[41] <https://developer.mozilla.org/en-US/docs/Web/API/HTMLCanvasElement/toBlob>. Acesso em 05/06/2017.

Para isso, o ImageJS provê um ambiente alternativo ao de produção – onde todos os recursos da plataforma são buscados a partir uma única origem (arquivos JS com todos as bibliotecas e códigos encapsulados). O Ambiente de desenvolvimento consiste utilizar códigos locais passivo de alteração pelo desenvolvedor em questão, isso permite que o desenvolvedor crie novos recursos para a plataforma.

Para isso, basta obter a biblioteca JS principal do ImageJS (leia o tópico Contribua, seção 7.2), vincular o conteúdo da plataforma em uma página HTML (aprenda a vincular lendo o tópico Importação da Plataforma, seção 6.2). Deve-se também informar ao editor que estará em ambiente de desenvolvimento, preenchendo a variável IMAGEJS_ENVIROMENT. Na Figura 14, pode-se observar como ficaria o código dessa definição.

```
<script>
    var IMAGEJS_ENVIROMENT = 'development';
</script>
```

Figura 14 – Como definir Ambiente de Desenvolvimento

Fonte: Autor.

Mediante a esta definição deve-se obter o código HTML e CSS presentes na biblioteca principal do ImageJS (biblioteca JS), e adicioná-los no corpo da página HTML em questão. Para facilitar tal processo a plataforma fornece um diretório chamado “development”, referente a todos os códigos desenvolvidos até o momento, estáticos, e escritos numa página HTML; mediante a isso, basta apenas trabalhar em um dos arquivos presentes neste diretório (cada arquivo representa uma estrutura diferente do editor, como a interface horizontal e vertical). Após o desenvolvedor concluir as implementações que desejava, precisa agora inserir os códigos HTML e CSS na biblioteca principal (para alterar códigos JS basta editar o arquivo principal diretamente). A Figura 15 demonstra onde deve ser sobrescrito esses valores.

```
/**...8 linhas */
this.html = {
    editions: '<editions> <edition name="new" onclick="imageJs.open()" t
    edition: '<edition> <img /> <input name="imagejs[{count}]" type="hid
    studio: '<studio> <input id="up-devise" type="file" hidden/> <tools-
};

/**...8 linhas */
this.css = '@import "https://fonts.googleapis.com/icon?family=Material+I
```

Figura 15 – Elementos Lógicos para Sobrescrita de HTML e CSS

Fonte: Autor.

5.6 ARQUITETURA DAS BIBLIOTECAS

Durante a descrição de todo este capítulo, percebe-se que para possibilitar a implementação funcional do ImageJS, foi necessário haver o envolvimento de vários conceitos e tecnologias trabalhando em simultâneo. Porém, cada biblioteca tecnológica não trabalha independente através de outro endereço computacional, mas sim, todos os recursos da plataforma estão unificadas em um único diretório, ou melhor, em um único arquivo.

Tendo em vista que os vários recursos da plataforma estão centralizados em um único arquivo, facilitando em grande escala e importação e vinculação da ferramenta, deve-se considerar que a organização desses recursos seja relevante à estrutura do projeto. Considerando que o HTML, CSS, e o JS possuem extensões de acesso divergentes entre si, centralizar todos seus dados num único arquivo de extensão unificada também tornou-se um desafio de arquitetura. A primeira consideração é que estas tecnologias não serão acessadas separadamente em seus arquivos, mas todo conteúdo será importado na página do sistema vinculante; portanto, a extensão dos arquivos já não é um problema, mas a organização lógica dos recursos sim. Pode-se observar na Figura 16 o modelo de arquitetura lógico de comunicação das bibliotecas.

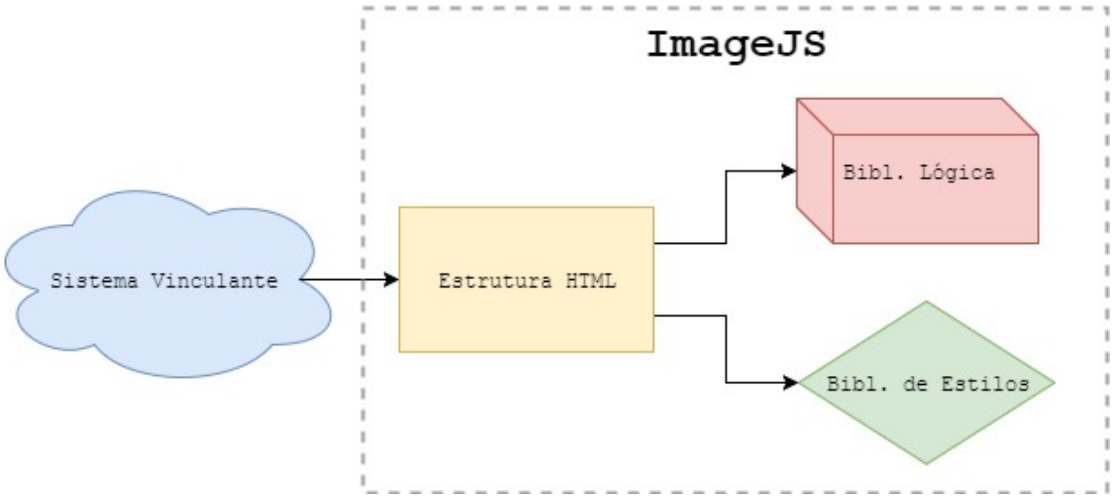

Figura 16 – ImageJS, Arquitetura de Comunicação

Fonte: Autor.

O modelo de comunicação consiste na maneira que as tecnologias se relacionam no escopo lógico da plataforma (chamadas de métodos e recursos); o modelo de armazenamento que os códigos estão salvos é representado pela arquitetura de física das bibliotecas, onde os *scripts* são centralizados em valores lógicos JavaScript, posteriormente essa linguagem dinâmica os insere devidamente no DOM do sistema vinculante. Na Figura 17, pode-se conferir a representação visual dessa arquitetura.

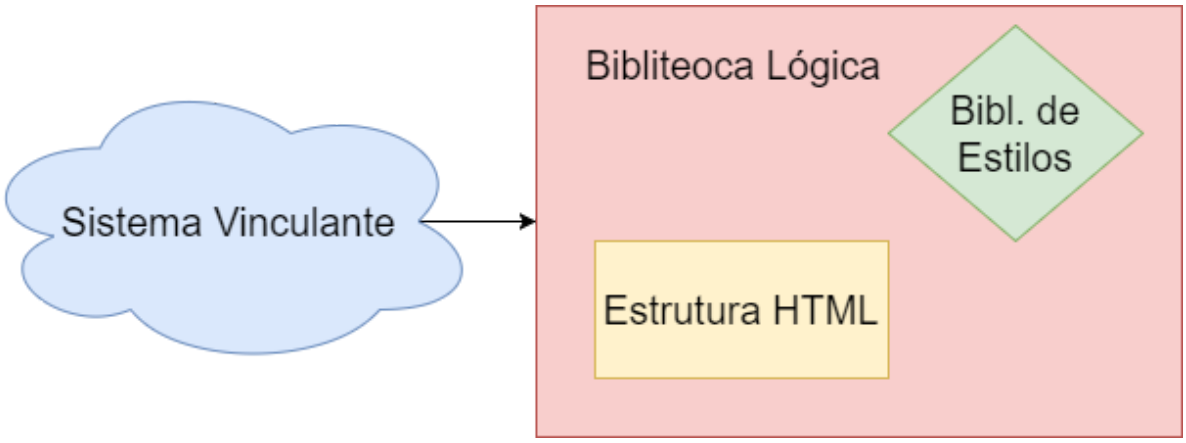

Figura 17 – ImageJS, Arquitetura Física das Bibliotecas

Fonte: Autor.

6 CENÁRIO DE UTILIZAÇÃO

Neste tópico será demonstrado o cenário de utilização da ferramenta, ou seja, os procedimentos necessários para vincular a plataforma bem como os passos básicos para se utilizar dos recursos da ferramenta.

6.1 TESTES

O escopo de desenvolvimento da plataforma é direcionado a recursos lógicos, com intuito de desenvolver um minieditor base para quantas ferramentas de edições forem necessárias. Devido a isso, os testes do projeto projeto são pouco práticos e visam exemplificar o funcionamento técnico da plataforma.

6.1.1 NAVEGADORES TESTADOS

Como um dos objetivos é aprimorar a habilidade de desenvolvimento *cross-browser*, testou-se a plataforma em 5 navegadores populares. Para isso, tentou-se testar apenas nas últimas versões disponíveis de cada navegador até o momento. A Tabela 7 lista os navegadores que foram convenientes para o escopo de testes do editor.

Tabela 7 – Lista de Navegadores Testados

Navegador	Versão
Mozilla Firefox	54
IE ou Internet Explorer	11

Google Chrome	59
Opera	10
Safari	6

6.1.2 COMPORTAMENTO DA PLATAFORMA

Os testes efetuados na plataforma fazem referência a velocidade que o estúdio é apresentado na tela para o usuário e o quão rápido a ferramenta aplica as manipulações no quadro de edição. Porém, o minieditor se comporta diferente em cada situação, pois seus códigos são interpretados totalmente no lado do cliente; o navegador e as capacitações técnicas do dispositivo utilizado pelo usuário implicam no quão bem a plataforma irá funcionar. Portanto não há como definir de maneira genérica como será o comportamento da ferramenta; mas tendo em vista sua estrutura lógica simples, pois trata-se de códigos básicos sem muitas estruturas de repetição e escrita em eventos do navegador, a plataforma não exigirá muito do dispositivo, a não ser que o sistema vinculante seja um sistema com vários recursos, comprometendo assim parte do processamento dedicado ao estúdio.

Considerando uma situação específica, a qualidade da imagem que será enviada para edição é a principal característica que pode afetar diretamente no desempenho. Os testes foram efetuados pelo autor em um dispositivo com processador Core i3 (primeira geração) e 6 Gigas de memória RAM. Nos 5 navegadores testados o editor apresentou lentidão semelhante quando uma imagem com qualidade superior a 1920x1080 (ou com conteúdo acima de 5MB) foi enviada para edição; devido a grande quantidade de pixels proporcionais e propriedades visuais que uma imagem desse porte possui, o minieditor utiliza mais recursos computacionais para armazenar e percorrer seus elementos.

6.1.3 RESULTADO DAS FERRAMENTAS DEMONSTRATIVAS

Além dos filtros citados no capítulo de desenvolvimento, com objetivo de testar a eficácia da plataforma e a capacidade de sustentar e aplicar quaisquer ferramentas de manipulação de imagem, apenas implementando seus respectivos algoritmos, listou-se a seguir os resultados de aplicação das ferramentas demonstrativas que o editor fornece até o momento.

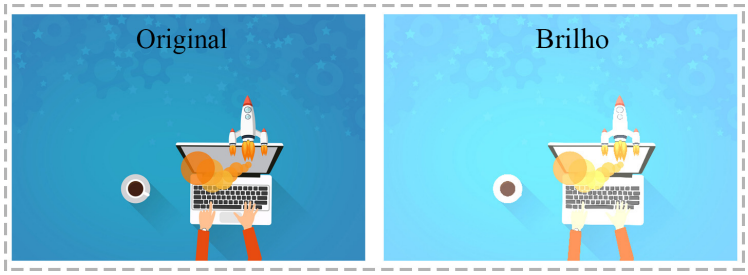

Figura 18 – ImageJS, Resultado da Ferramenta de Brilho

Figura 19 – ImageJS, Resultado da Ferramenta de Contraste

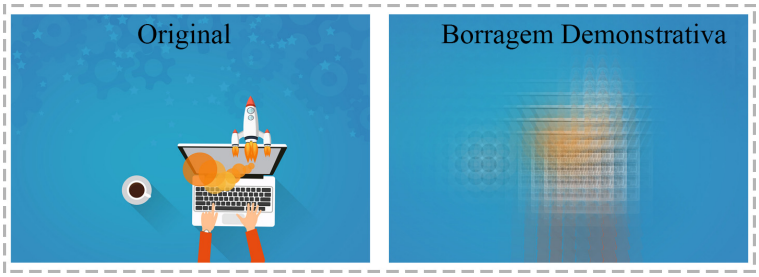

Figura 20 – ImageJS, Resultado da Ferramenta de Borragem Demonstrativa

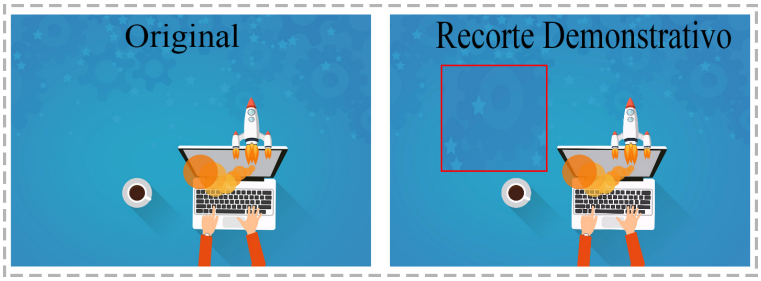

Figura 21 – ImageJS, Resultado da Ferramenta de Recorte Demonstrativo

6.2 IMPORTAÇÃO DA PLATAFORMA

O primeiro passo antes de se efetuar qualquer utilização de teste ou desenvolvimento com a plataforma proposta, deve-se anteriormente vincular seus códigos em uma página Web qualquer, carregando assim toda a estrutura do ImageJS, incluindo suas bibliotecas de estilos e lógicas. Para isso, basta obter o código centralizado em uma única biblioteca através do endereço Web de contribuição (endereço referenciado no tópico Contribua). É possível efetuar o *download* dos recursos e vinculá-los em uma página HTML a partir do seu próprio dispositivo, caso contrário, basta referenciar o diretório Web diretamente. A Figura 22 exemplifica a importação do código central do ImageJS a partir da Web.

```
<script
    type="text/javascript"
    src="https://github.com/ribaslucian/imagejs/blob/master/assets/image-js.js">
</script>
```

Figura 22 – Código de Vinculação a Partir da Web

Fonte: Autor.

Após essa vinculação dos arquivos, agora resta incorporar o código estrutural em qualquer lugar do corpo da página, de preferência em um formulário. Isso informará para biblioteca do ImageJS que é necessário iniciar os recursos na página no elemento em questão. A maneira que a incorporação da plataforma foi uma proposta diferencial já referenciada pela Figura 1.

6.3 INCORPORAÇÃO DE RECURSOS

Devido a facilidade de se implementar, a incorporação dos recursos foi proposto como diferencial do trabalho. Bastando apenas escrever a *tag* <image-js> em qualquer lugar da página do sistema vinculante, todos os recursos da plataforma estarão disponíveis a partir de tal referência. A Figura 1 exemplifica de maneira técnica a incorporação dos recursos dentro de um formulário Web, porém pode-se referenciar externamente a isso e utilizar a opção de baixar a imagem editada. O Tópico Simulando Ambiente Real, descreve um exemplo prático simulando uma situação real.

6.4 SIMULANDO AMBIENTE REAL

Para exemplificar a utilização da plataforma em ambiente real, optou-se por uma situação simples e bastante comum em sistema Web – O Cadastro de um usuário em um sistema qualquer. Neste exemplo, o usuário preencherá um formulário com Nome, E-mail e Senha; além desses campos textuais, será possível também enviar e editar a foto de perfil, esta opção será feita com o ImageJS, que está vinculado juntamente com o formulário. O Código do formulário de exemplo está escrito na Figura 33, Apêndice B.

Passo 01: Preencher como de costume os dados do formulário, conforme demonstra a Figura 23.

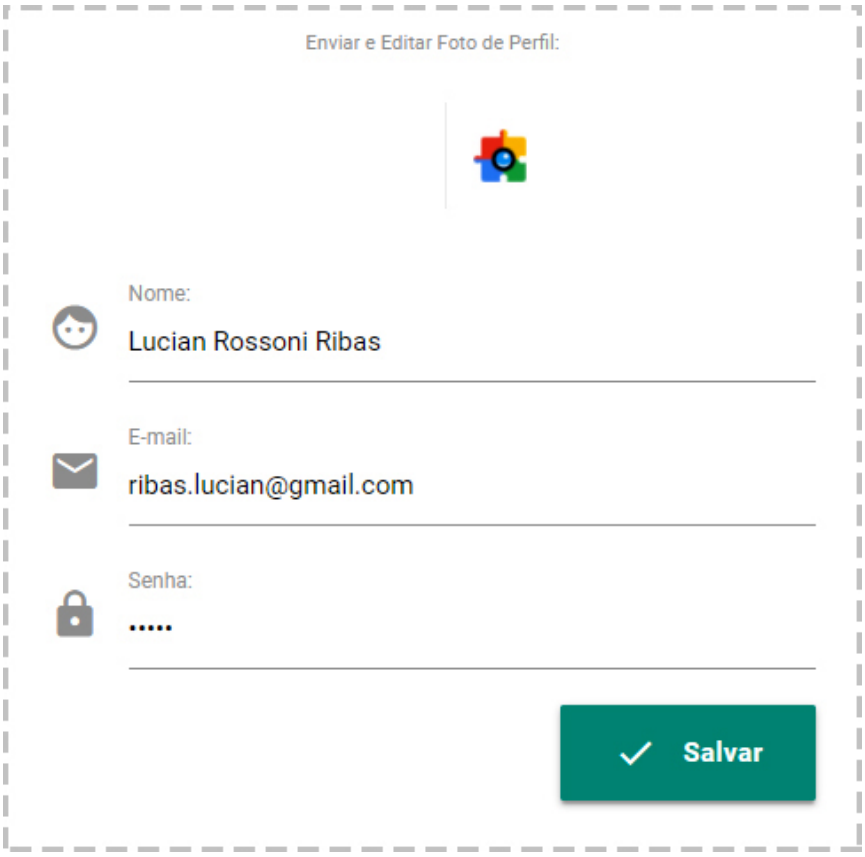

Figura 23 – Passo 01: Preenchimento do Formulário com o ImageJS

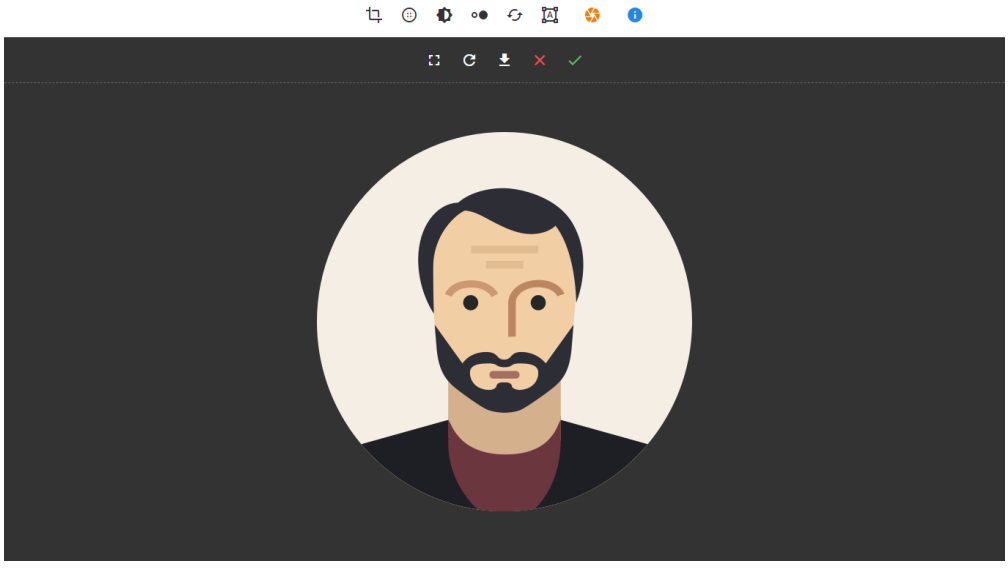

Figura 24 – Passo 02: Editar Imagem com o ImageJS

Passo 02: Enviar uma imagem qualquer para o perfil através da plataforma ImageJS, utilizando das ferramentas demonstrativas (brilho, contraste, filtros). Para isso basta

clicar no ícone representativo da ferramenta disponível no formulário, e seguir as instruções do editor para aplicar as edições a imagem.

Passo 03: Finalizar a edição clicando no botão verde do menu opções superior que possui o ícone de "*check*". Seguir os passos do editor para não iniciar uma nova edição.

Passo 04: Após finalizar a edição, o editor irá inserir no formulário uma miniatura representativa da imagem que o usuário editou. A Figura 24 demonstra isso. Agora resta apenas enviar os dados ao servidor. Para entender como implementar o recebimento das edições no ambiente do servidor, leia o tópico Escopo de Servidor.

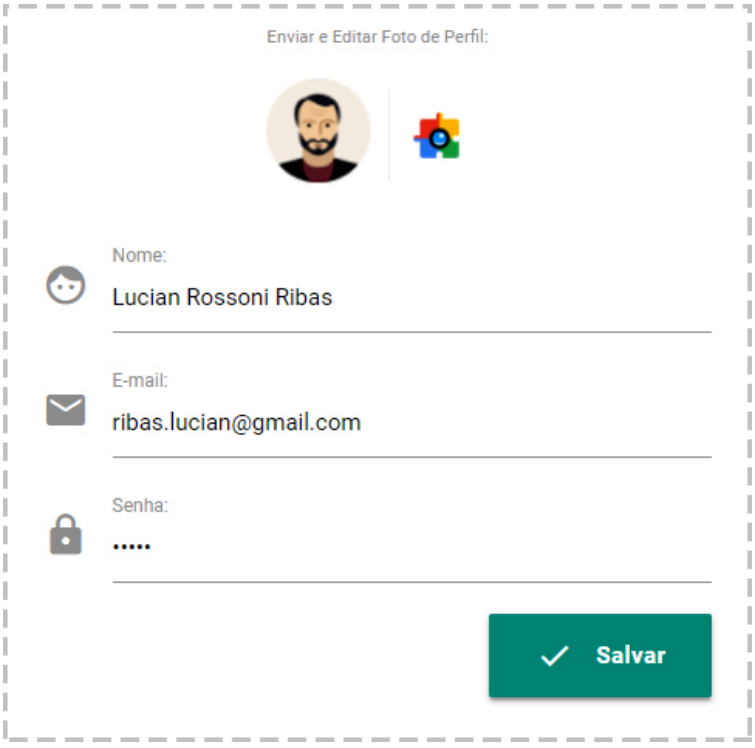

Figura 25 – Passo 04: Edição Efetuada e Formulário Preenchido

6.5 ESCOPO DE SERVIDOR

Um dos diferenciais propostos pelo trabalho em questão é a possibilidade de vincular a plataforma em um formulário HTML qualquer, ou seja, cada edição concluída resulta em um valor adicionado neste formulário. Os dados de formulários Web costumam ser enviados a um servidor, este por sua vez processa os dados gerando informações. Portanto, toda edição efetuada com o ImageJS será atribuída em um formulário, que será provavelmente enviada a um servidor Web, este deve gerar a informação respectiva a edição.

Como já descrito, toda edição efetuada com a plataforma proposta resulta em conteúdo textual encriptado com o algoritmo Base64; logo, quando as edições concluídas chegarem no servidor de destino, ele deve primeiramente tratar estes valores e não apenas salvar os dados das edições como se fossem arquivos (imagens) convencionais. Para isso, os tópicos seguintes demonstraram exemplos de como gerar a informação (o arquivo de imagem respectivo) das edições enviadas ao servidor. Considerando a instalação básica dos recursos, as linguagens de servidores PHP e Ruby provêm recursos nativos para decriptar e salvar o arquivo representativo de textos encriptados no formato Base64, dessa forma não será necessário de nenhuma biblioteca adicional.

6.5.1 EXEMPLO DE SERVIDOR EM PHP E RUBY

Para receber os dados das edições deve-se percorrer o índice “imagejs” que representa um Vetor armazenado na variável de ambiente referente ao envio referente do formulário, POST ou GET (variáveis $_GET e $_POST respectivamente). Recomenda-se que seja criado uma função pública para desempenhar o processo de conversão das edições textuais para arquivos.

Antes disso deve-se considerar que em PHP existe uma variável de ambiente chamada $_FILES, referente aos arquivos enviados de outras origens. Recomenda-se que durante o processo de conversão não armazene as imagens em memória diretamente, mas salve-as nessa variável de ambiente, possibilitando assim que outros escopos do servidor em

questão também utilize os dados das edições, já convertidas em arquivos de imagem. A Figura 39 (Apêndice E), demonstra uma sugestão de código estruturado para receber as edições do ImageJS em PHP.

Em ruby (utilizando o Rails ou o Sinatra) a sugestão de implementação se assemelha a de PHP, porém, com algumas instruções divergentes, pois as variáveis para receber os dados enviados de outras origens (Post e Get) não são as mesmas; nesse caso, deverá ser percorrido a variável “params” no índice “imagejs”. A Figura 40 (Apêndice E), demonstra uma sugestão de código estruturado para receber as edições do ImageJS em Ruby.

6.5 O EDITOR ONLINE

O editor *online* consiste em um endereço Web específico para utilizar e aplicar testes prévios sobre a plataforma proposta, neste endereço o estúdio já está vinculado na página inicial e basta utilizá-lo, porém não há envio para o servidor, pode-se apenas aplicar testes dos recursos *front-end* que referenciam diretamente o projeto; pode-se também verificar como a plataforma se comportaria caso vinculado em um site. Previamente, para ter acesso a todo o diretório de arquivos do ImageJS *online*, pode-se utilizar este[42] endereço; internamente a isso, pode acessar o arquivo “development/horizontal.html”, que representa o último modelo de desenvolvimento adotado.

6.6 CRIE EDIÇÕES COM A PLATAFORMA

Tendo em vista a complexibilidade para criar uma manipulação de imagem em âmbito Web, pois é necessário manipular e entender recursos de HTML (o elemento Canvas), CSS (como no trabalho proposto utilizou-se para proporções), e JS (para manipular os contextos e criar o algoritmo da manipulação). O ImageJS abstraiu o processo de manipulação e disponibilizou um método público que permite que o desenvolvedor escreva algoritmos proprietários para edição da imagem de maneira simples e direta, sem preocupação com

[42] <http://lucian.hol.es/imagejs/>. Disponível a partir de 05/06/2017 por tempo indeterminado.

elementos estruturais, visuais e até mesmo lógicos (interação com o canvas está transparente), basta referenciar a função e criar a ferramenta de manipulação de acordo com a própria criatividade e necessidade. Para isso, basta ter o estúdio vinculado e incorporado na página HTML em questão.

O “write” recebe como parâmetro uma função anônima, função está que por sua vez recebe um único parâmetro que representa os pixels ou elementos da imagem vinculada com o estúdio no momento de referência de “write”. O corpo da função permite que se percorra os pixels e defina novos valores, o retorno é “void”, ou seja não há necessidade de retorno. A Figura 26 demonstra a documentação e como pode ser feito a utilização do método “write”.

```
<script>
    /**
     * Método simplista para criar algotimos de manipulação do
     * canvas referente a imagem vinculada no editor no momento.
     *
     * MY_CUSTOM_VALUE é uma variável representativa do valor
     * que se deseja inserir na cor do pixel em questão, altere isso.
     *
     * @param {function} _callable: function(pixels)
     * @return void
     */
    imageJS.write(function (pixels) {
        // percorre-se todos os pixeis do Canvas
        for (var i = 0; i < pixels.length; i += 4) {
            pixels[i] = MY_CUSTOM_VALUE;   // R| valor vermelho do pixel
            pixels[i+1] = MY_CUSTOM_VALUE; // G| valor verde do pixel
            pixels[i+2] = MY_CUSTOM_VALUE; // B| valor azul do pixel
        }
    });
</script>
```

Figura 26 – Crie Edições Personalizadas com o Método Write

Fonte: Autor.

7 CONSIDERAÇÕES FINAIS

O desenvolvimento dessa plataforma exigiu bastante familiaridade com as tecnologias Web *front-end* (HTML, CSS e JS), principalmente para estruturar soluções que não utilizassem apenas de capacitações da linguagem em questão, mas possibilitar o entrosamento dos recursos, resultando assim em códigos mais simples e descritivos.

Devido ao escopo de aplicabilidade da ferramenta, para nomeá-la considerou-se os principais pilares do desenvolvimento: a união das palavras Imagem e Javascript, sendo representado então por ImageJS, unificado em uma única palavra para representação mais simples.

O ImageJS ainda não se encontra na sua primeira versão pois o desenvolvimento dos recursos básicos ainda não foram concluídos. Pois das ferramentas definidas na seção 4.1.1, foi desenvolvido até o momento 5 filtros personalizados, a ferramentas de brilho e contraste, e parte das ferramentas de recorte e borragem. Para a plataforma ficar pronta para utilização, primeiro deve-se concluir o desenvolvimento das ferramentas propostas e então finalizar a versão 1 do editor.

7.1 DESENVOLVIMENTOS FUTUROS

A plataforma proposta permite que continuamente novos recursos sejam criados de acordo com a necessidade e evolução das tecnologias empregadas. Porém, ainda falta algumas implementações referentes as ferramentas listadas anteriormente pelo tópico de Definição das Ferramentas, tais como:

- **Ferramenta de recorte:** Resta desenvolver o retângulo translúcido que permite que o usuário recorte a imagem livremente, arrastando e redimensionando o retângulo com mouse.

- **Ferramenta de Borragem:** Centralizar a borragem num espaço relativo ao redor do pixel que o usuário clicar.
- **Ferramentas textuais:** Nenhuma ferramenta textual foi desenvolvida logicamente, apenas seus componentes gráficos.
- **Padronizar Tamanho do Canvas:** O tamanho do canvas de edição atualmente é de 3000 pixels proporcionais (horizontal e vertical). Qualquer imagem enviar será redimensionada de maneira escalável a estas dimensões. Deve-se implementar um comportamento que o canvas se adapte a imagem, e não o contrário como é atualmente.
- **Limitar Número de Edições:** O editor ainda não permite que o desenvolvedor defina limitações da quantidade de edições que o usuário pode efetuar. No envio de fotos para o perfil do usuário, geralmente seleciona-se apenas uma imagem, nessa situação seria interessante que a plataforma tivesse essa utilidade.

Há pendente também recursos pós edição, como implementações para criar interações com imagens já editadas pelo usuário, permitindo que abra novamente uma edição anteriormente concluída, efetue *download* e aplique novas edições; o cancelamento de edições concluídas também não foi desenvolvido.

Desconsiderando implementações pendentes do ImageJS, o editor fornece recursos estruturais para criar novas ferramentas, então, qualquer desenvolvedor interessado pode criar novas ferramentas para enriquecer a plataforma, seja para aplicar implementações *front-end* ou para treinar técnicas em processamento manipulações de imagem.

De acordo com o desenvolver do recurso é possível que com a contribuição da comunidade de desenvolvedores (que se interessem pela proposta), a plataforma considerada atualmente como um minieditor acabe se tornando um estúdio para aplicação de técnicas em específicas e manipulação de imagens de maneira mais complexa. Para isso basta ler o tópico Contribua.

7.2 CONTRIBUA

Ferramentas implementadas e mantidas apenas por um único desenvolvedor não tendem a crescer, é necessário que a comunidade interessada também desenvolva e proponha novas ideias referentes a plataforma em questão.

Através da tecnologia Git, fornecida pelo Github, o ImageJS foi centralizado neste endereço Web[43], que consiste em uma plataforma online que permite o desenvolvimento em grupos, controlando também as versões de cada arquivo. Então, caso deseje ter acesso às bibliotecas atualizadas pelo editor, efetue o *download*, pode-se também enviar novos códigos e ideias, basta utilizar dos recursos do Git e contribuir gratuitamente com o desenvolver dessa ferramenta.

7.3 CONCLUSÃO

Desenvolver recursos para o ambiente Web é geralmente motivante, talvez pela possibilidade de alcance do recurso, pela presença da Web no dia-a-dia, pela capacidade técnica que exige tais escopos de desenvolvimento, ou apenas pelo gosto pessoal do desenvolvedor.

Muita vezes se detém de um pré objetivo teoricamente simples, assim como o do trabalho proposto que seria criar ferramentas para edição de imagens; mas como a descrição do trabalho deixa claro, o grande empenho foi no desenvolvimento entorno da plataforma, e não diretamente na implementação das ferramentas de edição de imagem. Devido a isso, o desenvolvimento das ferramentas de edição acabou tornando-se segundo plano, onde o primário se tornou a implementação da própria estrutura para se criar as edições; isso explica o desenvolvimento vago de tal escopo, e o termo “demonstrativo” utilizado quando descrito sobre as ferramentas da plataforma.

[43] <https://github.com/ribaslucian/imagejs>. Disponível a partir de 05/06/2017 por tempo indeterminado.

REFERÊNCIAS

GONZALEZ, Rafael; WOODS, Richard. **Processamento Digital de Imagens**. 3. ed. São Paulo: Pearson Education, p. 1. 2010.

TAURION, Cezar. **Computação em Nuvem: Transformando o Mundo da Tecnologia da Informação**. Rio de Janeiro: BRASPORT Livros e Multimídia LTDA, p.17. 2009.

FAYAD, M. E. et al. **Object-oriented Application frameworks**. Communications of the ACM, Vol. 40, 10 p., 1997.

MAZZA, L. **HTML5 e CSS3: Domine a web do futuro**. Casa do Código, 2013. ISBN 9788566250053. Disponível em: <http://www.casadocodigo.com.br/products/livro-html-css>.

SCHEID, Felippe. **Fundamentos de CSS: Criando Design para sistemas web**. 1. ed. Foz do Iguaçu: Editora Outbox Interativa, 2015.

RAMALHO, J. A. **Curso completo para desenvolvimento web**. Rio de Janeiro: Elsevier, 2005.

REMOALDO, P. **O Guia prático do dreamweaver CS3 com PHP, JavaScript e Ajax**. 1. ed. Lisboa: Centro Atlântico, 2008.

RUTTER, J. **Smashing jQuery interatividade avançada com JavaScript simples**. Porto Alegre: Bookman Companhia Editora Ltda, 2012.

SILVA, M. S. **Desenvolva aplicações web profissionais com uso dos poderosos recursos de estilização das CSS3**. São Paulo: Novatec Editora, 2012.

MICARONI, Felipe; BUENO, Felippe; ZACHARIAS, Guilherme. **Evolução da Programação Web**. 2008. p.34. Trabalho de Conclusão de Curso – Ciência da Computação, Faculdade Comunitária de Campinas. Campinas, 2008.

MARABESI, Matheus; DOUGLAS, Michael. **Zend Certified Engineer: Descomplicando a certificação PHP**. Disponível em <https://www.casadocodigo.com.br/pages/sumario-certificacao-php>. Acesso em: 06 jun. 2017.

HSM Educação Executiva. **O essencial de 2016 – conteúdos da melhor revista de gestão e liderança do Brasil: para alavancar seu desempenho pessoa, o da sua equipe e sua empresa**. 1. ed. São Paulo: Casa Educação Soluções Educacionais LTDA, 2017.

TORRES, Joaquim. **Zend Certified Engineer: Descomplicando a certificação PHP**. Disponível em <https://www.casadocodigo.com.br/products/livro-gestao-produtos>. Acesso em: 06 jun. 2017.

SABIN-WILSON, Lisa. **WordPress Web Design for Dummies**. 2. ed. Hoboken: John Wiley & Sons, p.76. 2013.

SANTAELLA, Lucia. **Comunicação ubíqua: Repercussões na cultura e na educação**. São Paulo: Paulus. 2014.

CRANE, D. et al. **Ajax em ação**. 1. ed. Londres: Manning Publications, 2006.

GUGONI, Marcel. **Migrar para a nuvem ajuda a economizar até 90% nos gastos de TI**. Disponível em: <http://www.amcham.com.br/business-in-growth/noticias/migrar-para-a-nuvem-ajuda-a-economizar-ate-90-nos-gastos-de-ti>. Acesso em: 11 set. 2015.

LUCA, Cristina. **Buscas no Google: 15% das pesquisas diárias são novas, nunca foram feitas.** Disponível em: <http://idgnow.com.br/internet/2013/09/25/busca-google-15-das-pesquisas-diarias-nunca-foram-feitas-antes/>. Acesso em: 05 set. 2015.

MACHADO, ANDRÉ. **Por ano, 125 bilhões de imagens são compartilhadas na rede**. Disponível em: <https://oglobo.globo.com/sociedade/tecnologia/por-ano-125-bilhoes-de-imagens-sao-compartilhadas-na-rede-8301345>. Acesso em: 14 mai. 201.

Material Design. Disponível em: <https://www.w3schools.com/w3css/w3css_material.asp>. Acesso em 22/05/2017.

APÊNDICE A - Ícone Representativo e Telas do Sistema

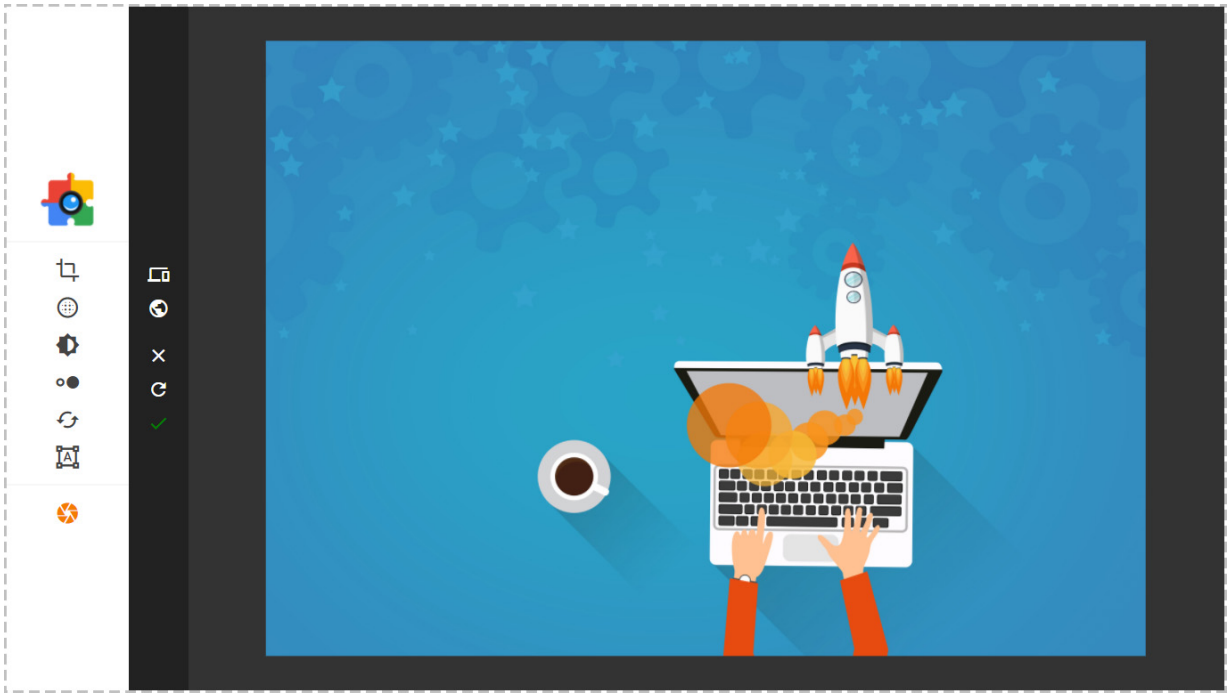

Figura 27 – ImageJS, Interface Vertical

Fonte: Autor

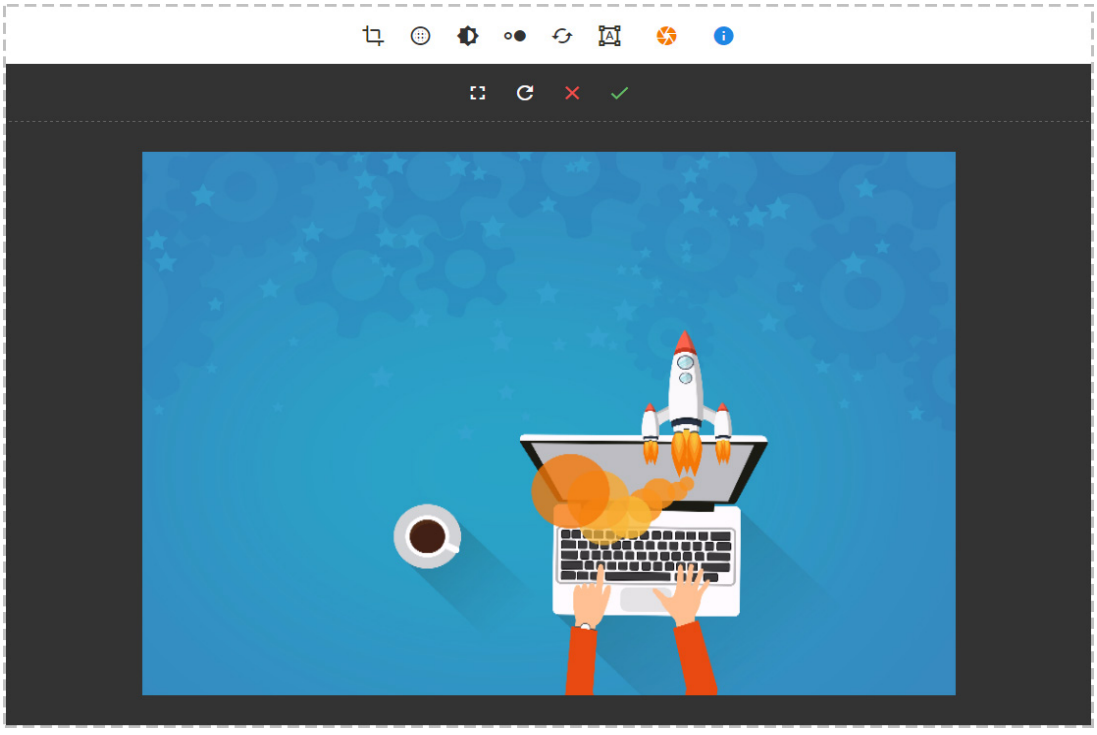

Figura 28 – ImageJS, Interface Horizontal o Modelo Adotado

Fonte: Autor.

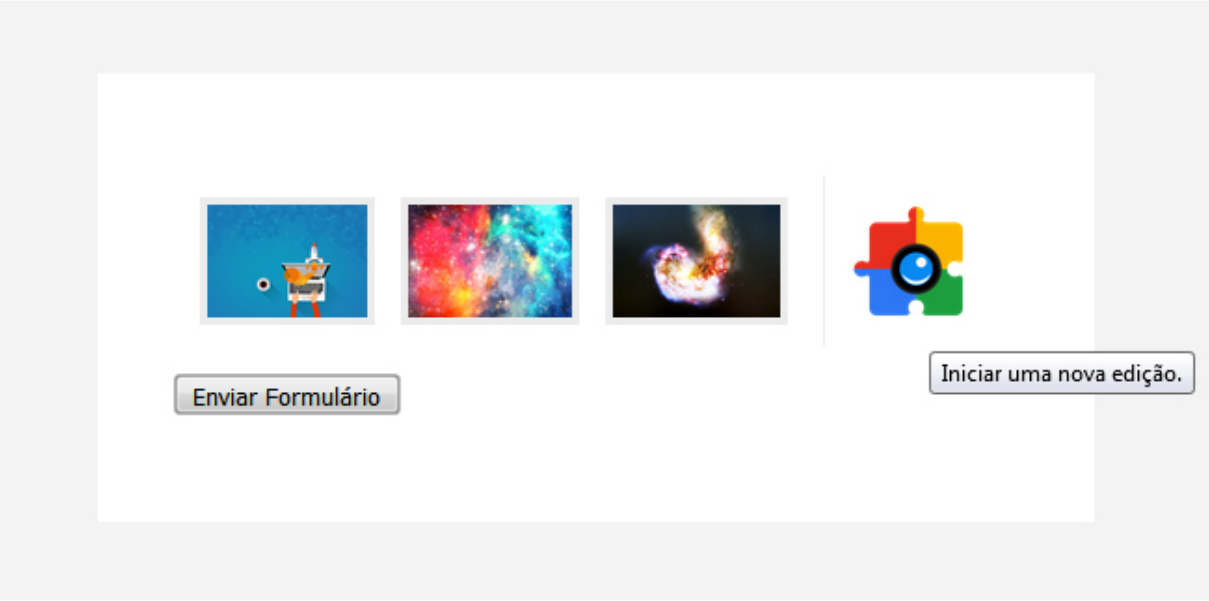

Figura 29 – ImageJS, Início de Edição e Edições Concluídas

Fonte: Autor.

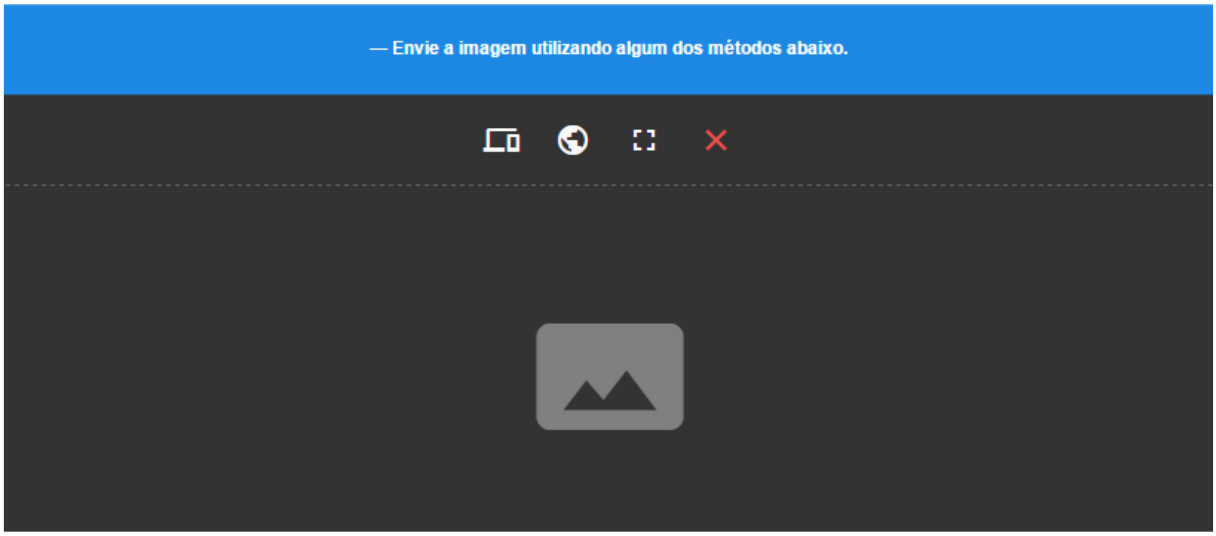

Figura 30 – ImageJS, Apresentação de Mensagens e Ferramentas Secundárias

Fonte: Autor.

APÊNDICE B - Soluções Estruturais

```
<!-- 1 ESTRUTURA DE VINCULAÇÃO -->
<image-js>

    <!-- 1.1 RECIPIENTE DE EDIÇÕES -->
    <editions>

        <!-- 1.1.1 REPRESENTANTE LÓGICO DE EDIÇÃO -->
        <edition>

            <!--
            Para cada edição concluída, será incrementando +1 no valor
            numérico no subtexto "{count}" contido no atributo "alt" e
            "name" (da tag <img> e <input> respectivamente).
            -->
            <img src="{diretorio_temporario_da_imagem}"
                 alt="{count}° Edição efetuada com o estúdio ImageJS." />

            <input name="imagejs_edition_{count}"
                   type="file" />
        </edition>

        <!-- 1.1.2 INICIALIZADOR -->
        <edition name="new" title="Iniciar uma nova edição.">
            <img src="{diretorio_online_do_icone_representativo}"
                 alt="Ícone Representativo do estúdio ImageJS." />
        </edition>
    </editions>
</image-js>
```

Figura 31 – Estrutura HTML de Vinculação

Fonte: Autor.

```
<!-- 2 ESTÚDIO -->
<studio>

    <!--
    2.0.1 RECIPIENTE VINCULANTE PELO DISPOSITIVO
    Input que receberá a imagem enviada através do dispositivo do usuário.
    -->
    <input type="file" imagejs-hidden />

    <!-- 2.1 RECIPIENTE PRIMÁRIO  -->
    <tools-primary>

        <!-- 2.1.1 GRUPOS DE FERRAMENTAS PRIMÁRIAS -->
        <tools name="primary">

            <!-- 2.1.1.1 FERRAMENTA DE RECORTE -->
            <tool name="crop" title="Ferramenta de recorte.">
                <i class="material-icons">&#xE3BE;</i>
            </tool>

            <!-- 2.0.2 OMISSÃO DE REDUNDÂNCIA - FERRAMENTAS -->
        </tools>

        <!-- 2.0.3 OMISSÃO DE REDUNDÂNCIA - GRUPOS DE FERRAMENTAS -->

        <!-- 2.1.2 MENSAGENS -->
        <info name="success" class="imagejs-bg-green">

            <!-- 2.1.2.1 CENTRALIZADOR -->
            <centralizer>
                ✓   Edição concluída com sucesso.
            </centralizer>
        </info>

        <!-- 2.0.4 OMISSÃO DE REDUNDÂNCIA - MESANGENS E INFORMAÇÕES -->
    </tools-primary>

    <!-- 2.2 RECIPIENTE SECUNDÁRIO -->
    <tools-secondary>
        <!-- 2.0.5 OMISSÃO DE REDUNDÂNCIA - GRUPOS DE FERRAMENTAS SECUNDÁRIAS -->
    </tools-secondary>

    <!-- 2.3 RECIPIENTE DE EDIÇÃO -->
    <editing>

        <!-- 2.3.1 SAGUÃO DE EDIÇÃO -->
        <container>
            <!-- 2.0.6 CANVAS - O QUADRO DE EDIÇÃO -->
            <canvas width="5000" height="5000"></canvas>

            <!-- 2.3.1.2 SELETOR TRANSLÚCIDO -->
            <selector></selector>
        </container>

        <!-- 2.0.7 IMAGEM OCULTA PROJETORA DE DIMENSÕES -->
        <img />
    </editing>
</studio>
```

Figura 32 – Estrutura HTML de Edição

Fonte: Autor.

```
<form method="post" action="../server/receive.php">
    <div class="center">
        <small class="grey-text">
            Enviar e Editar Foto de Perfil:
        </small>

        <image-js></image-js>
    </div>

    <br/>
    <div class="input-field">
        <i class="material-icons prefix grey-text">&#xE87C;</i>
        <input placeholder="Nome Completo" id="name" type="text">
        <label for="name">Nome:</label>
    </div>

    <div class="input-field">
        <i class="material-icons prefix grey-text">&#xE0BE;</i>
        <input placeholder="exemplo@exemplo.com" id="email" type="emai
        <label for="email">E-mail:</label>
    </div>

    <div class="input-field">
        <i class="material-icons prefix grey-text">&#xE897;</i>
        <input placeholder="******" id="password" type="password">
        <label for="password">Senha:</label>
    </div>

    <button class="waves-effect waves-light btn btn-large teal right"
        <i class="material-icons left">&#xE876;</i>
        Salvar
    </button>
</form>
```

Figura 33 – Código Completo do Formulário de Exemplo

Fonte: Autor.

APÊNDICE C - Soluções de Estilo

```
/* [...] */

/**
 * Elementos que serão totalmente
 * centralizados em função dos seus recipientes.
 */
studio info,
studio info text,
studio tools,
studio container,
studio selector {
    left: 50%;
    top: 50%;
    position: absolute;
    transform: translate(-50%, -50%);
}

/**
 * Elementos que devem se comportar como colunas.
 */
studio tools-primary,
studio tools-secondary,
studio editing {
    position: relative;
    float: left;
    width: 50%;
    height: 100%;
}

/**
 * Elementos que devem iniciar ocultos.
 */
studio,
studio editing img,
studio tools-primary tools,
studio canvas,
studio selector,
studio tools-secondary tools,
studio tools-primary info,
studio [name="editing"] {
    display: none;
}

/* [...] */
```

Figura 34 – Trecho de Estilização Centralizado em Tags

Fonte: Autor.

```css
/**
 * Elementos que serão totalmente
 * centralizados em função dos seus recipientes.
 */
studio info,
studio info text,
studio tools,
studio container,
studio selector {
    left: 50%;
    top: 50%;
    position: absolute;
    transform: translate(-50%, -50%);
}
```

Figura 35 – Exemplo de Posicionamento utilizando Atributo Transform

Fonte: Autor.

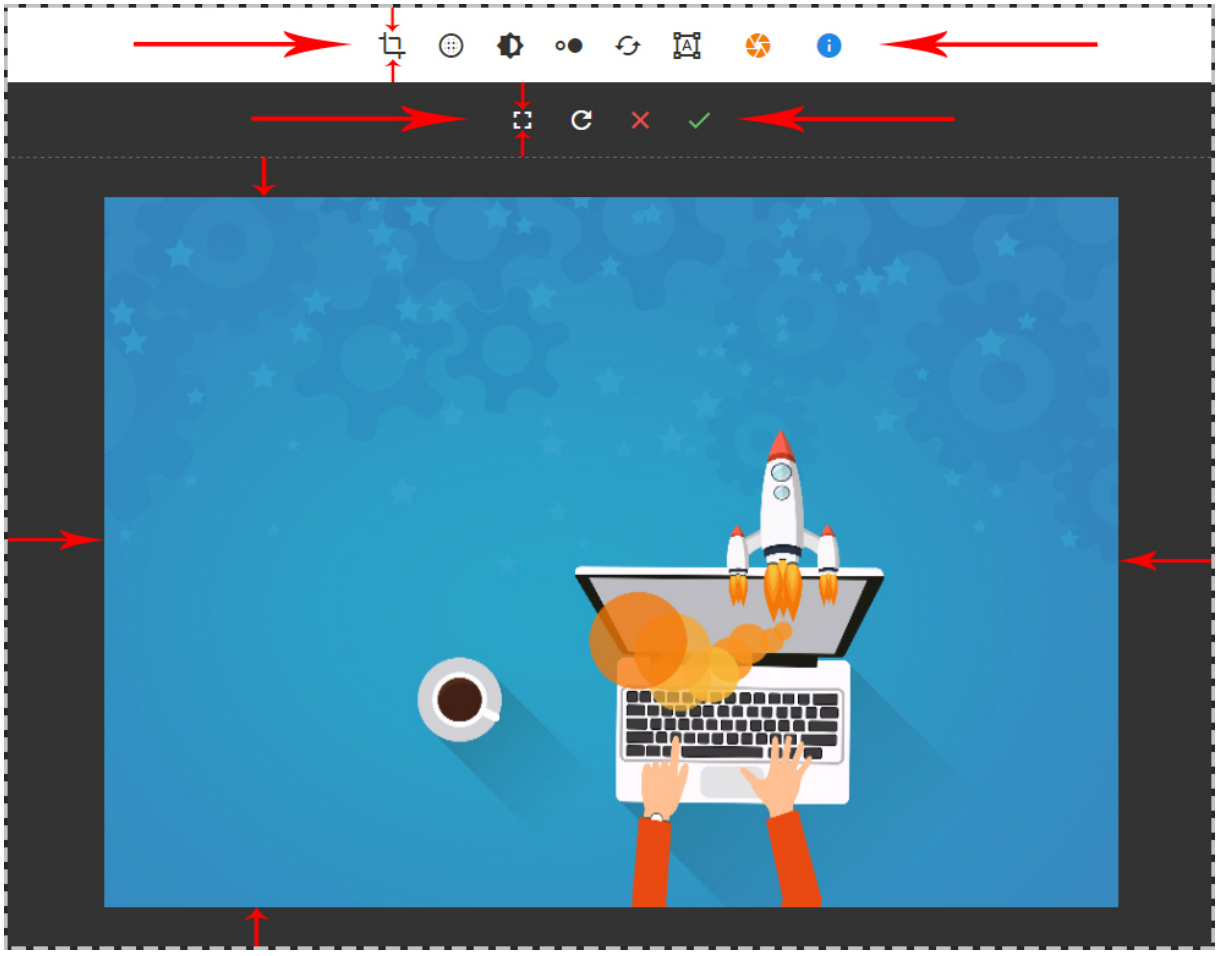

Figura 36 – Resultado de Centralização dos Elementos

Fonte: Autor.

APÊNDICE D - Soluções Lógicas

```
/*
 * Codificação lógica do ImageJs disposta por função
 * construtora para melhor referênciação dos seus métodos.
 */
var ImageJs = function () {...453 linhas };

// instanciação manual do ImageJs para informar que desejamos trabalhar com
// seus recursos; estes por sua vez são controlados automaticamente pelo
// conteúdo instanciado do objeto.
var imageJs = new ImageJs();

// iniciamos os recursos básicos
imageJs.writeDOMComponents();
```

Figura 37 – ImageJS, Modelo Lógico de Referência

Fonte: Autor.

```
imageJs.editor.context.drawImage(
    image,

    // coordenada X e Y do canvas que o
    // resultado do recorte será posicionado
    0, 0,

    // largura e altura do imagem
    // resultado do recorte dentro do canvas
    image.width, image.height,

    // coordenadas X e Y iniciais do
    // sub-retângulo de recorte da imagem
    0, 0,

    // largura e altura do sub-retângulo de recorte
    imageJs.$canvas.get(0).width, imageJs.$canvas.get(0).height
);
```

Figura 38 – Escrita da Imagem

Fonte: Autor.

APÊNDICE E - Exemplos Server-Side para Recebimento das Edições

```php
<?php

/**
 * Intercepta do envio e converte os valores Base64 das edições
 * para sua origem que são as imagens editadas; logo em seguida,
 * armazena-as na variável de ambiente $_FILES, que representa
 * os arquivos enviados.
 *
 * @return void
 */
function imageJsMiddleware() {
    foreach ($_POST['imagejs'] as $editionNumber => $editionBase64) {
        $editionBase64 = substr(explode(';', $editionBase64)[1], 7);
        $_FILES['imagejs'][$editionNumber] = base64_decode($editionBase64);
    }
}

// referenciamos o middleware do ImageJs.
imageJsMiddleware();

// percorremos as edições efetuadas com o ImageJS.
foreach ($_FILES['imagejs'] as $edition => $image) {
    // salvamos a edição em questão no diretório e com o nome desejado.
    file_put_contents("../assets/editions/imagejs_$edition.png", $image);
}

die('Edições salvas');
```

Figura 39 – Exemplo PHP para recebimento as Edições

Fonte: Autor.

```ruby
# criamos um método para converter as edições para arquivos
def image_js
    params[:imagejs].each do |edition_id, edition_base64|
        params[:imagejs][edition_id] = Base64.decode64(edition_base64)
    end
end

# chamamos o método middleware do ImageJS
image_js()

# percorremos as edições enviadas
params[:imagejs].each do |edition_id, edited_image|
    # salvamos o arquivo
    File.open('../assets/imagejs_#{edition_id}.png', 'wb') do |file|
      file.write(edited_image)
    end
end
```

Figura 40 – Exemplo Ruby para Recebimento as Edições

Fonte: Autor.