Redesenvolvimento da Ferramenta EdiMM através do Framework JavaScript React

Daniel Ribeiro Soares Silva[1], André Constantino da Silva[1,2], Flavia Linhalis[2]

[1]Grupo de Pesquisa Mobilidade e Novas Tecnologias de Interação
Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas
Instituto Federal de Educação, Ciência e Tecnologia de São Paulo (IFSP)
Campus Hortolândia – SP – Brasil

[2] Núcleo de Informática Aplicada à Educação – UNICAMP
Rua Seis de Agosto, 50, Reitoria V - Cidade Universitária, Campinas – SP – Brasil

daniel.r@aluno.ifsp.edu.br, andre.constantino@ifsp.edu.br, flalin@unicamp.br

Abstract. *The Editor de Texto Multissemiótico e Multimodal (EdiMM) is a tool to build text in many semiosis, that is, involving images, sounds, videos, written and typed words in a unique document. EdiMM's user interface is considered to be Single Pages, which consists of buttons to access different tools and a blank canvas that is used for the insertion of the different semiosis. Therefore we believe that the application can take a lot of benefit from React, due to its high code reuse, components and, mainly, buttons and its presentation is concentrated in a single page.*

Resumo. *O Editor de Texto Multissemiótico e Multimodal (EdiMM) é uma ferramenta para construir textos com diversas semioses, ou seja, que envolvam imagens, sons, vídeos, palavras escritas e digitadas, em um único documento. A interface de usuário do EdiMM é constituída de uma página única, consistente de botões para o acesso das ferramentas disponíveis e uma tela branca onde são inseridas e manipuladas as diferentes semioses. Portanto acreditamos que a aplicação pode se beneficiar do uso do React, devido ao seu alto reuso de componentes, principalmente os botões e sua apresentação ser concentrada em apenas uma página.*

1. Introdução

Apesar de ser considerado um *framework* de JavaScript, React é uma biblioteca *open-source* de JavaScript para desenvolver grandes e complexas interfaces de usuário. Esta biblioteca é comumente utilizada para construir aplicações de página única (mais conhecidas pelo termo em inglês *Single-Page Application*) e também pela sua velocidade e praticidade de reutilização de código, já que oferece manutenção mais rápida e eficaz. Além disso, devido ao seu ponto principal ser a programação funcional, o React pode ser usado para a criação de aplicações móveis através do React Native, tanto para Android quanto para IOS.

Devido à grande evolução da internet, novas tecnologias e formas de solucionar problemas foram surgindo. O uso de JQuery está em declínio quando comparado com novos *frameworks* e bibliotecas, seu uso tem se tornado menos necessário devido às atualizações recentes do JavaScript. O React, ao invés de utilizar o JavaScript puro, utiliza o JSX. É um arquivo JavaScript que permite o uso de HTML no seu código interno para renderizar componentes e subcomponentes. O uso de JSX é opcional e não é obrigatório para usar o React.

A partir das qualidades citadas do *framework* React foi definido o objetivo deste trabalho. Sua finalidade é atualizar a ferramenta EdiMM, de maneira que seu código passe a utilizar tecnologias mais atuais e com constante crescimento e suporte da comunidade, já que as tecnologias utilizadas na versão original se encontram defasadas e em constante declínio de uso e também, de maneira que essa atualização de tecnologia facilite possíveis manutenções futuras.

Neste artigo serão abordados diferentes tópicos relacionados ao desenvolvimento do EdiMM desde problemas na sua versão original até o seu redesenvolvimento através do React.

Primeiramente será apresentado o funcionamento da ferramenta, sua interface e funcionalidades (Seção 2), em tópicos posteriores serão mencionados quais são outras ferramentas que possuem certas similaridades (Seção 3), a metodologia usada (Seção 4), como foi executado o desenvolvimento da nova versão utilizando React (Seção 5) e um comparativo entre linhas de código das diferentes versões (Seção 6). Este artigo se encerra com conclusões na Seção 7.

2. A Ferramenta EdiMM

O Editor de Texto Multissemiótico e Multimodal (EdiMM) é uma ferramenta para construir textos que envolvam imagens, sons, vídeos, palavras escritas e digitadas, em um único documento. Também é uma ferramenta de interação multimodal, ou seja, deve suportar diferentes modalidades de entrada (como teclado, toque, mouse e caneta) para acionar comandos, adaptando as possibilidades de entrada conforme disponíveis no dispositivo usado. O desenvolvimento do EdiMM, iniciado em 2015, é incremental e evolutivo, no qual diversos trabalhos de TCC e de Iniciação Científica contribuem com uma característica e, consequentemente, a evolução da ferramenta. Apesar dos diversos trabalhos realizados no EdiMM, este está utilizando versões defasadas de tecnologias.

A Figura 1 apresenta a interface do EdiMM, ou seja, a sua página principal. É uma tela em branco onde podem ser inseridas as semioses e as ferramentas disponíveis dispostas em um menu na lateral esquerda cujos atributos podem ser modificados no menu superior, o qual se altera conforme a ferramenta escolhida.

Figura 1. Interface principal do EdiMM visualizado em um computador de mesa.

A interface de usuário do EdiMM é construída através de uma página única, consistente de diversos botões com diversas funcionalidades. Na barra superior estão todos os botões que controlam as funções de manipulação de texto, tais como alteração de cor, fonte e espessura de linha de desenho, essas alterações são feitas de forma geral, afetando somente as próximas inserções. As ferramentas laterais são as principais semioses do sistema (desenhar, apagar, incluir título, caixa de texto, incluir formas geométricas e incluir arquivos), após selecionar uma determinada função, o usuário pode incluir o que deseja em toda a parte branca da tela.

O sistema também pode ser utilizado em diferentes tamanhos de telas devido à sua responsividade, estabelecida no arquivos de estilização. Por exemplo, a Figura 2 demonstra a ferramenta EdiMM sendo acessada por meio de um dispositivo móvel.

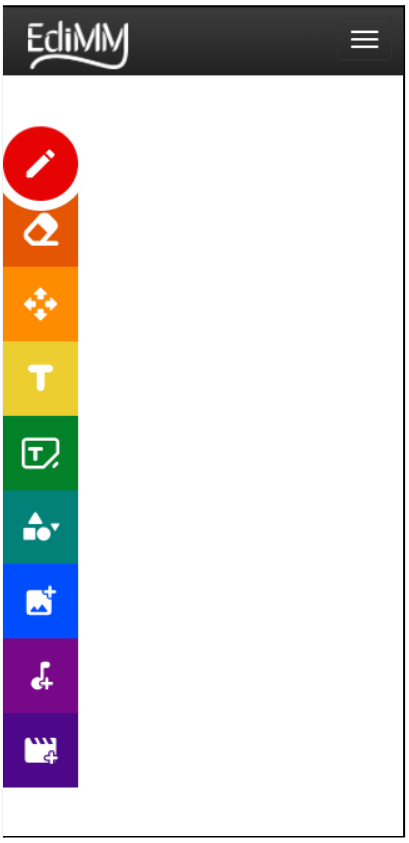


Figura 2. Visão do sistema em um dispositivo móvel de tamanho de tela menor

O EdiMM foi desenvolvido orientado a eventos, sendo assim, rotinas responsáveis por capturar estes eventos e executar determinados comportamento conforme o evento selecionado. À medida que o usuário escolhe uma funcionalidade na barra lateral de ferramentas, os eventos de entrada de dados se comportam de forma diferente. Por exemplo, ao escolher a ferramenta desenho à mão livre é habilitada no canvas a possibilidade de desenho, registrando a posição do ponteiro e pintando na tela com a cor determinada, e quando selecionada a funcionalidade de caixa de texto é inserido um objeto editável na tela, que efetua a leitura de cada tecla pressionada pelo usuário.

3. Trabalhos Correlatos

O Google Jamboard (Figura 3) é a ferramenta da Google produzida para criação de diferentes semioses, ela é utilizada juntamente da Google Workspace (antigo G Suite, serviço que oferece versões de vários produtos Google). Foi lançado em 2017 juntamente com seu *hardware* para uso comercial. Suas funcionalidades são parecidas com as do EdiMM, possibilitando a entrada de diferentes semioses e oferecendo a possibilidade do trabalho síncrono como seu ponto principal. Seu código é restrito por possuir *hardware* próprio.

O Excalidraw (Figura 4) é uma ferramenta de desenho para navegadores que possibilita a criação de desenhos, inserção de texto e o uso de formas geométricas. O sistema possui uma funcionalidade de camadas onde é possível personalizar quais semioses tem prioridade de aparição na tela, além de possibilitar a edição de cada uma delas, podendo alterar o tamanho, cor, espessura e até mesmo opacidade.

A ferramenta Excalidraw foi criada utilizando React e Typescript e possui um repositório com código aberto, tendo suas funcionalidades separadas em componentes e possibilitando seu uso como biblioteca em outros projetos, além de possuir uma configuração de linguagem, onde é verificado o idioma do navegador para determinar qual será aplicado no sistema.

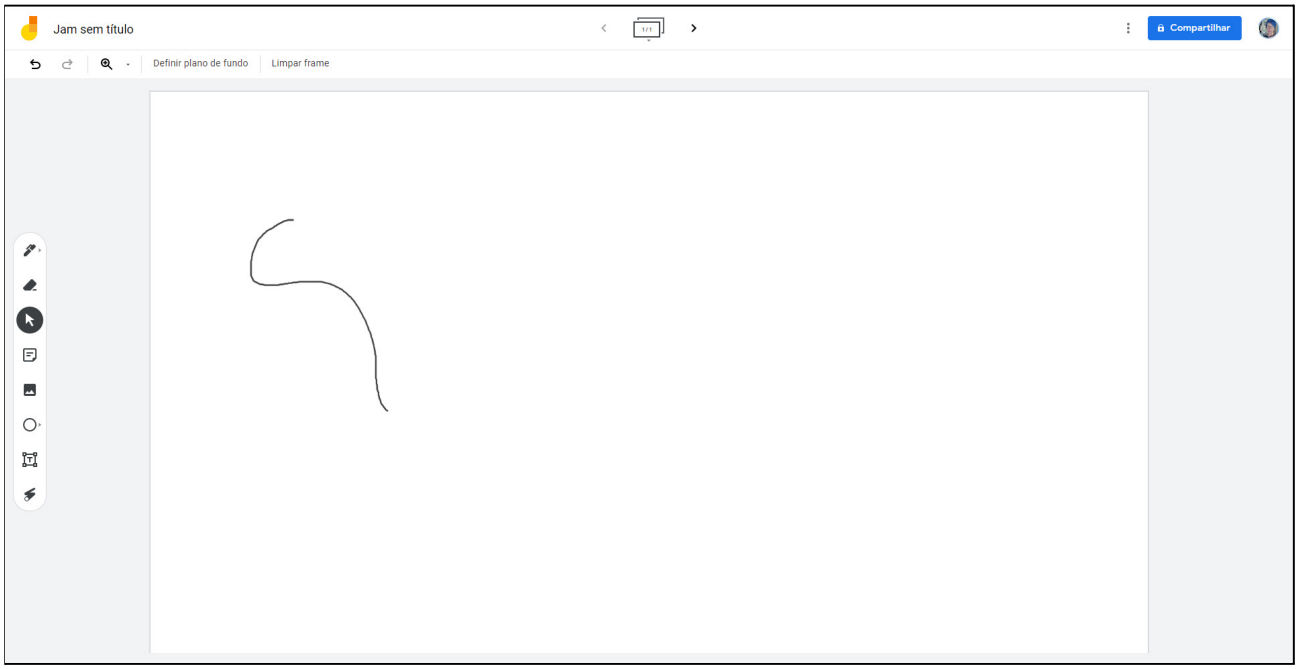


Figura 3. Interface do Google Jamboard

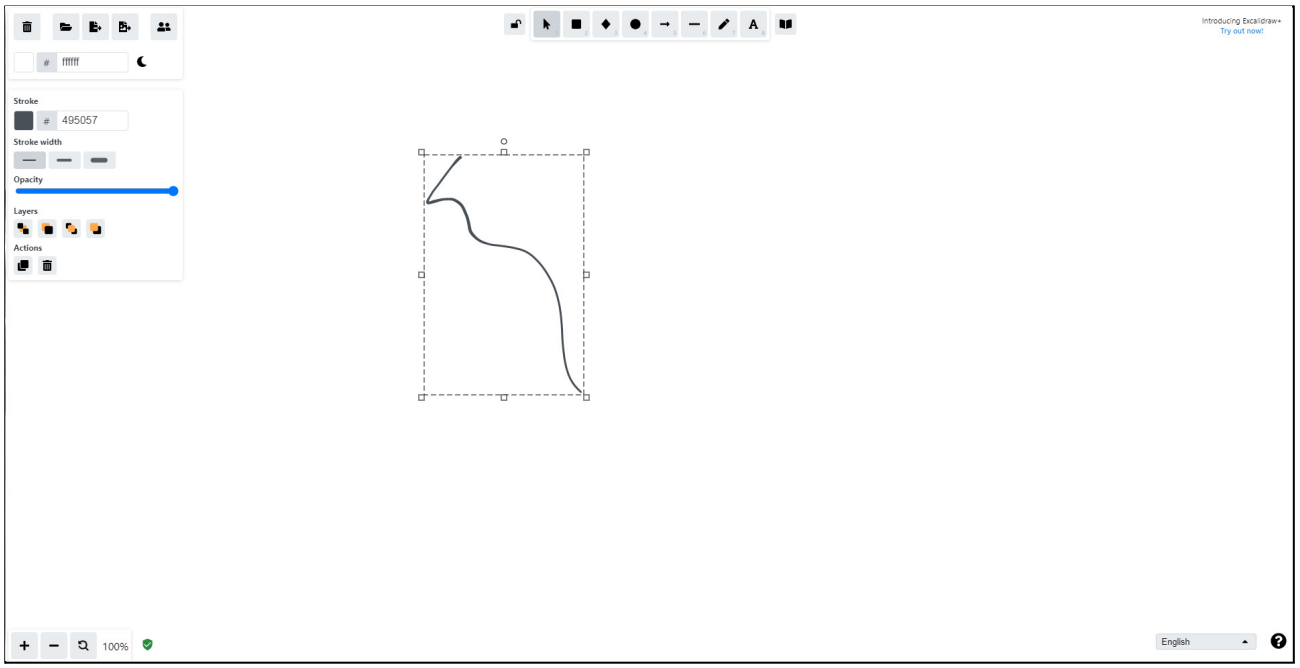


Figura 4. Interface do Excalidraw

Freite *et al*. (2015) apresenta um levantamento de funcionalidades de diferentes aplicativos disponíveis para a edição de textos, considerando também o sistema operacional que suporta o aplicativo; esta foi a base para a construção do EdiMM. A Tabela 1 expande a tabela proposta por Vascon *et al*. incluindo o Google Jamboard, o Excalidraw e o EdiMM.

Tabela 1. Sistema Operacional, Funcionalidades e Mídias presentes em aplicações para dispositivos móveis - expandido com base no trabalho de Vascon *et al (2015)*.

APLICATIVOS	SIST. OPER.				FUNCIONALIDADES E MÍDIAS								
	App IOS	App Android	Windows	Web	Texto	Imagem	Fotografia	Anexar áudio	Gravar áudio	Anexar vídeo	Gravar vídeo	Link	Desenho livre
FiiNote		✓			✓	✓	✓	✓	✓	✓	✓	✓	✓
Evernote	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓
Google Keep		✓		✓	✓	✓	✓	✓	✓			✓	
One Note	✓	✓	✓	✓	✓	✓	✓		✓			✓	✓
Skitech	✓	✓	✓		✓	✓	✓						✓
MyScript Smart Note	✓	✓			✓	✓	✓						✓
Squid		✓	✓		✓	✓	✓						✓
Genial Writing2	✓	✓			✓	✓	✓						
Write		✓	✓			✓	✓						✓
Bloco Notas +		✓			✓	✓							✓
Lecture Notes		✓			✓								✓
Excalidraw				✓	✓	✓							✓
Google Jamboard		✓		✓	✓	✓	✓						✓
EdiMM	✓	✓		✓	✓	✓		✓		✓			✓
ReactEdiMM				✓	✓	✓		✓		✓			✓

4. Metodologia

De acordo com Sommerville (2018), “o desenvolvimento de *software* não é interrompido quando o sistema é entregue, mas continua por toda a vida útil do sistema”. A evolução de *software* se trata de etapas que um sistema passa até ser entregue, elas podem ser diversificadas, dependem principalmente do tipo de sistema e do tipo da solicitação de mudança a ser efetuada. A solicitação pode possuir diferentes objetivos.

Neste projeto será utilizado o método exploratório devido à ausência de outros métodos que auxiliem a identificação de soluções no contexto apresentado pelo projeto. De início, é fundamental compreender as principais funcionalidades que o EdiMM possui para que o re-desenvolvimento proposto possa ser efetuado com clareza e eficácia. A principal função da migração de tecnologia será facilitar o entendimento dos códigos da ferramenta, facilitar a resolução de *bugs*, além de reduzir o acoplamento nos arquivos JavaScript.

O escopo considerado para o redesenvolvimento do EdiMM é um *Proof of Concept* (POC -

prova de conceito), no qual é a evidência de que um software pode ser bem ou mal sucedido e também serve para possivelmente identificar erros técnicos que possam interferir no funcionamento e nos resultados esperados.

5. Desenvolvimento

O desenvolvimento do novo EdiMM foi feito através de entregas de partes do código empregando componentes e funcionalidades, utilizando a ferramenta original como base para desenvolvimento da atual, mantendo suas principais funcionalidades como desenhar, adicionar arquivos, apagar arquivos, adicionar textos/caixas de texto, alterar cor, tamanho de texto, fonte, deixar em itálico, negrito e sublinhado, alterar espessura de linha e salvar em pdf. O código, seguindo a filosofia do projeto em ser aberto, se encontra disponível em repositório na Internet (https://github.com/danielrssilva/ReactEdiMM) A primeira coisa que faz o React ser tão popular é a apresentação de um código mais fácil de ser compreendido, dado que sua organização é feita em componentes e subcomponentes que compõem uma página e também por sua forte integração com JavaScript.

React foi escolhido para o redesenvolvimento da ferramenta devido à comunidade *open-source* de React que vem crescendo cada dia mais, além de apresentar um grande número de projetos desenvolvidos no mercado de trabalho, demonstrando o crescimento do *framework*.

Com isso, teve-se a ideia de adaptar a ferramenta EdiMM, que é construída em tecnologias já defasadas (JQuery e Bootstrap versão 3.3.6, sendo a atual versão 5.1) através do *framework* React. Isso possibilitará uma melhor organização e manutenção do código, já que todos os componentes se encontram na mesma página e também são muito reutilizados. O projeto visa uma redução de linhas de código, já que alguns arquivos do EdiMM podem chegar a quase 10.000 linhas de código, dificultando a escalabilidade e a manutenção do sistema.

O jQuery é uma estrutura JavaScript amplamente usada para desenvolvimento de *front-end* da web. No entanto, jQuery tende a fazer muitas coisas que o JavaScript puro já faz. Além disso, o gerenciamento de estado ainda não está pronto, o que ReactJs já resolve como seu recurso principal.

O autor Khuat Tung (2018) cita em seu artigo, *Developing a frontend application using ReactJs and Redux*, que o React permite que os usuários dividam a interface de usuário em partes independentes e reutilizáveis chamadas *React Components*. Os componentes React implementam um método de renderização que obtém dados de entrada e retorna o que exibir. Cada componente tem vários métodos de ciclo de vida (são invocados em diferentes fases e esses ciclos de vida têm o objetivo de destruir componentes que não estão sendo utilizados) que podem ser substituídos para executar o código em momentos específicos durante o processo. Os métodos podem ser chamados usando a API Reacts. Essa peculiaridade também se aplica ao EdiMM, já que seu código é composto por uma única página com componentes iguais que desempenham funções diferentes. Essa organização tem um grande impacto na manutenção, na escalabilidade e também no desenvolvimento do mesmo, já que é organizado de uma forma mais coesa todo seu conteúdo.

Além disso, de acordo com Tung (2018), em vez de separar tecnologias, o React usa unidades fracamente acopladas chamadas de componentes. Cada componente é responsável por sua renderização e seu controle de estados, e através do JSX, permite um código com todos os tipos de arquivos utilizados para o desenvolvimento de uma página web (HTML, Javascript e CSS). Também permite que o React mostre erros e mensagens de aviso de forma mais clara. Com isso, o EdiMM se beneficia dessas peculiaridades do React, pois seu código original é muito complexo e extenso para manutenção.

5.1. Métodos de ciclo de vida do React

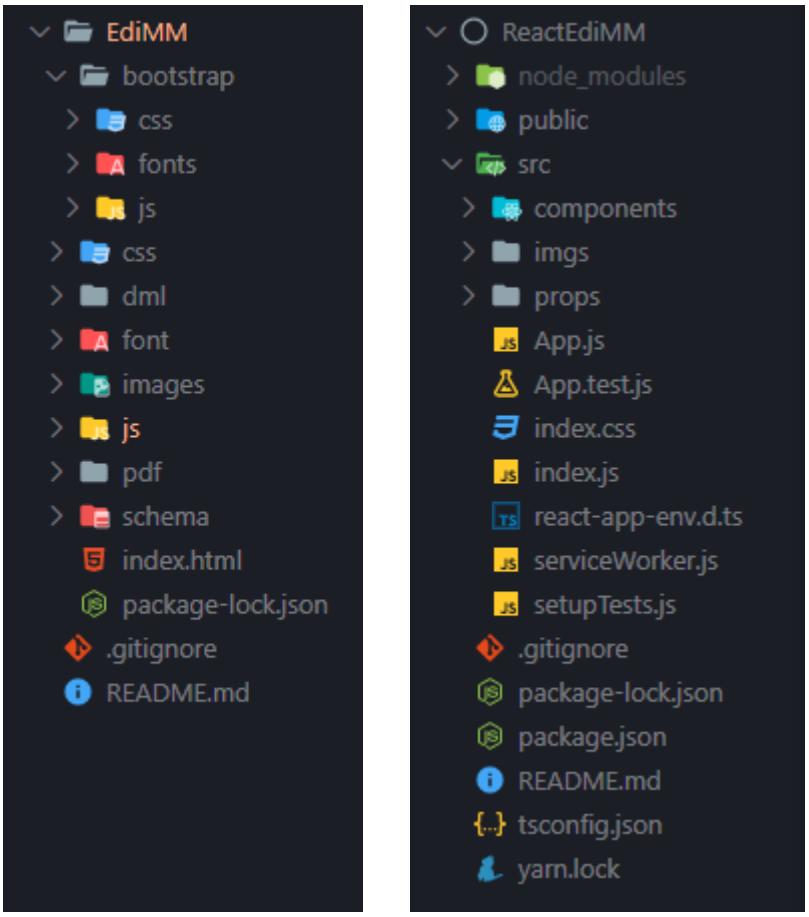
Existem três categorias de métodos de ciclo de vida: montagem, atualização e desmontagem.

Um componente é considerado montado quando é renderizado pela primeira vez, que é quando os métodos de ciclo de vida de montagem são chamados. A primeira vez que uma instância de componente é renderizada, ela não é atualizada. Começando com a segunda renderização, um componente é atualizado toda vez que é renderizado.

O período de desmontagem de um componente ocorre quando o componente é removido do DOM (*Document Object Model* - é a interface que representa como os documentos HTML e XML são lidos pelo seu browser). Isso pode acontecer se o DOM for renderizado novamente sem o componente ou se o usuário navegar para um site diferente ou fechar o navegador da web.

5.2. Alteração da organização do projeto

A organização dos arquivos da ferramenta (Figura 5b) foi feita através de padronizações de projeto, para que alterações futuras e o entendimento do projeto sejam mais rápidos. Foi utilizada a organização de arquivos recomendada pela documentação do React, pois ela é separada através da pasta de componentes com seus respectivos contextos. Essa organização foi considerada devido à grande quantidade de componentes presentes no EdiMM, de maneira que a ferramenta se beneficia de tal organização para possíveis futuras manutenções.



(a) (b)

Figura 5. Diferença da organização das pastas do sistema (a) original e (b) nova versão

Na versão original do EdiMM as pastas eram organizadas por tipos de arquivos (Figura 5a), na qual a pasta css possuía arquivos de estilização, a js armazenava todos os arquivos de código JavaScript.

A nova organização de pastas, conforme Figura 5b, foi feita da seguinte forma: a pasta *node_modules* contém todos os módulos necessários para que seja executada a aplicação em máquina. Esta pasta é gerada através do package.json que é um arquivo que controla as versões e

quais são as bibliotecas utilizadas no projeto. A pasta *public* é a pasta de arquivos públicos, automaticamente gerada pelo ReactJs, que contém o index.html e o ícone da aplicação. A pasta *src* possui toda a organização dos arquivos do sistema, na qual a pasta *components* tem o código de cada um dos componentes utilizados na ferramenta (Figura 6), *imgs* possui arquivos de imagem que são utilizados no visual da aplicação, a pasta *props* contém arquivos que exportam linhas de código que são muito repetitivas, tais como caminhos de arquivos de imagens.

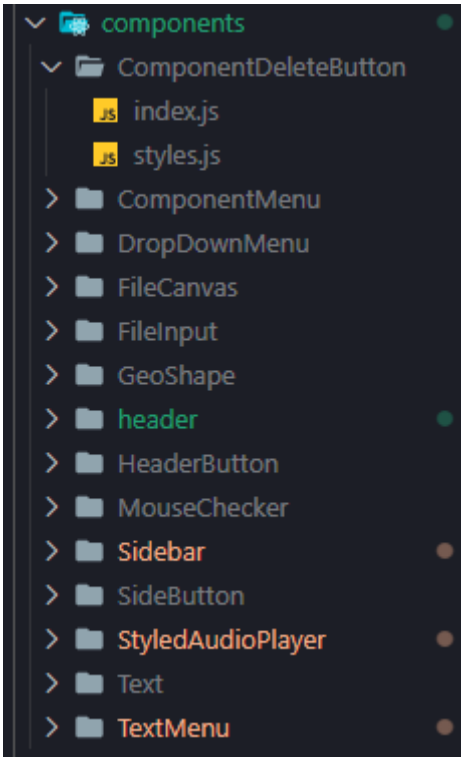


Figura 6. Demonstração da pasta de *components* com todas as subpastas de cada um dos componentes

Essa nova organização facilita na manutenção, devido à separação de cada um dos componentes. Caso um determinado componente apresente problemas, será fácil identificar onde ajustar o código, diferente do sistema anterior, que possuía um código muito acoplado e com muitas linhas.

A substituição da utilização da biblioteca de estilização de página Bootstrap ocorreu devido à grande customização presente na interface da ferramenta EdiMM, e suprindo a necessidade de personalização de cada elemento, a biblioteca *styled-components* se mostrou promissora devido à sua implementação.

Essa biblioteca é utilizada para criar componentes com estilização padrão, esses arquivos são exportados e utilizados em outros códigos como componentes HTML. Esses componentes visuais suportam parâmetros que podem ser utilizados dentro do próprio CSS para determinar alguma condição de renderização. Por fim, a biblioteca transforma todo o código em uma classe CSS que é aplicada para cada um dos componentes.

O código exposto na Figura 7 demonstra a criação de um componente que é chamado de *Container*. O mesmo recebe dois parâmetros (x e y) que representam onde será a localização desse determinado componente na tela.

Além desta biblioteca possibilitar o uso de variáveis dentro do arquivo de estilização, que é utilizado para definir a posição, cor, fonte, espessura etc. de cada componente, todo seu conteúdo é escrito em JavaScript ao invés de CSS. Sua função principal é a exportação de componentes com

estilos padronizados, que podem ser reutilizados em qualquer lugar do código após a importação.

A Figura 8 demonstra como é renderizado o botão de remoção de imagens. O botão, de fundo vermelho com ícone de uma lixeira e linhas de cor branca, é renderizado 35 pixels à esquerda e ao topo da imagem inserida.

```
export const Container = styled.div`
  position: absolute;
  left: ${({ x }) => `${x - 35}px`};
  top: ${({ y }) => `${y}px`};
  z-index: 1000;
`;
```

Figura 7. Exemplificação do uso da biblioteca *styled-components*

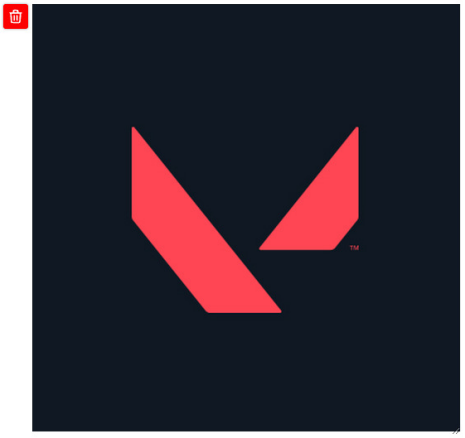


Figura 8. Exemplificação do resultado da renderização do componente *Container*

5.3. A nova interface de usuário do EdiMM

A nova interface do EdiMM (Figuras 9 e 10) possui uma nova renderização da barra superior e um novo agrupamento das ferramentas laterais. As funcionalidades de personalização da barra superior eram anteriormente renderizadas de acordo com a semiose inserida, porém na nova versão do EdiMM todos os botões se mantêm renderizados na tela independente de qual será a inserção do usuário, facilitando a localização dos mesmos dentre as diferentes formas de personalização. Além da alteração da barra superior, na barra lateral, os botões de inserção de diferentes tipos de arquivo foram unificados para melhor usabilidade.

JQuery não é mais necessário devido às novas versões do JavaScript. React utiliza componentização, deixando o código mais separado e com melhor manutenibilidade.

O menu superior controla os detalhes das semioses a serem inseridas através de alterações de estado, como por exemplo a cor da linha e sua espessura, já o menu lateral controla qual semiose será inserida após o clique do usuário na parte branca da tela.

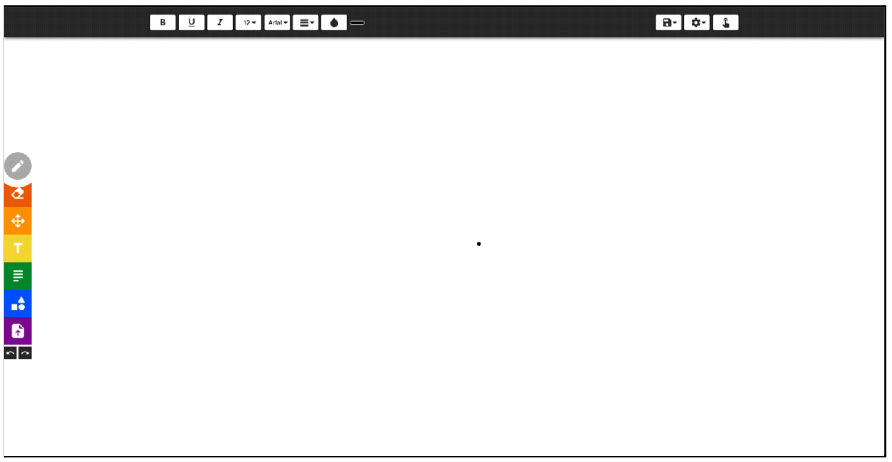


Figura 9. Interface do EdiMM

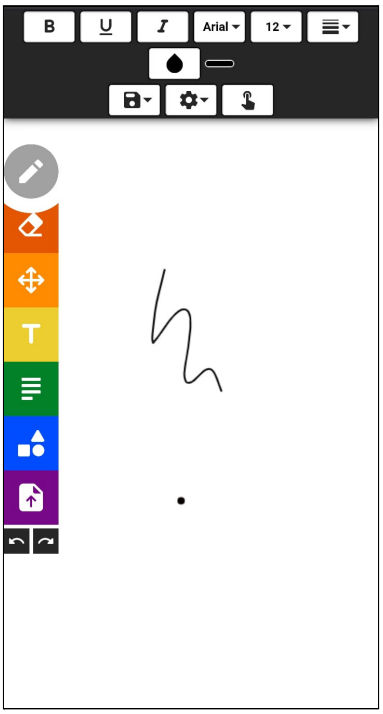


Figura 10. Interface do EdiMM em dispositivos móveis

No desenvolvimento da ferramenta de desenho foi utilizada a biblioteca React Canvas Draw - além de possuir personalização de cor e espessura da linha, possibilita o desenho livre na tela. Essa biblioteca foi encontrada através do site npmjs.com que possui várias bibliotecas que podem ser utilizadas junto com o ReactJs.

A Figura 11 demonstra a utilização da biblioteca, quais são os parâmetros necessários para a renderização do componente e a sobreposição do estilo do do canvas.

```
<CanvasDraw
  brushColor={color}
  brushRadius={brushRadius}
  disabled={selectedFunction === "draw" ? false : true}
  lazyRadius={0}
  style={{
    postion: "fixed",
    width: "100vw",
    height: "100vh",
    top: "0",
    background: "#ffffffff",
  }}
  hideGrid
/>
```

Figura 11. Exemplo da utilização da biblioteca React Canvas Draw

5.5. Modificação da funcionalidade caixa de texto

O uso de estados do React determina os dados de cada componente, tais como quais atributos que serão renderizados em tela, e no caso do EdiMM, temos como exemplo a cor do texto, a fonte, o tamanho da fonte etc. Essa manipulação de estados é verificada toda vez que um valor específico é alterado, fazendo com que os componentes que utilizam essas determinadas variáveis sejam unitariamente re-renderizados. Desta forma, após o usuário inserir uma semiose na tela, a mesma tem seu contexto travado, de maneira que uma nova funcionalidade escolhida não irá influenciar em semioses já inseridas na tela.

Com isso foi possível implementar a funcionalidade de alterar configurações de texto, cor, fonte e tamanho para cada caixa de texto individualmente (Figura 9), além de possibilitar a deleção, como demonstra a Figura 12.

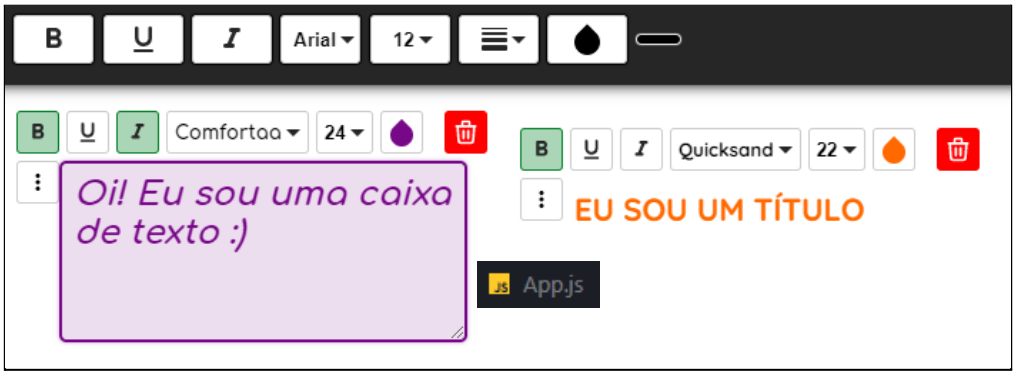


Figura 12. Menu de opções na caixa de texto e título

5.6. Modificação da funcionalidade apagar

Como descrito, devido à essa limitação dos estados do framework foi necessário implementar uma nova maneira de remoção de componentes inseridos pelo usuário, de maneira que cada componente possui um botão de menu com opções de alteração para aquele determinado componente (Figura 14).

ComponentDeleteButton > index.js

```
import react from "react";
import { Container, Button, Delete } from "./styles";

const ComponentDeleteButton = ({ y, x, color, handleDelete }) => {
  return (
    <Container x={x} y={y}>
      <Button
        onClick={() => handleDelete(x, y + 60)}
        color={color}
      >
        <Delete />
      </Button>
    </Container>
  );
};

export default ComponentDeleteButton;
```

Figura 13. Função que retorna botão de deleção

Figura 14. Botão de alternância do menu de uma caixa de texto

Cada componente possui em seu menu de opções um botão de deletar (Figura 15) que é individual e irá excluir a semiose de uma lista de todas que foram inseridas pelo usuário, cada uma com seu identificador (determinado pelo tipo de semiose inserido e sua localização na tela), cujo código é mostrado na Figura 16.

Figura 15. Botão de remoção de semiose de uma caixa de texto

App.js

```
const handleDelete = (x, y) => {
  let newCanvasContent = canvasContent;
  newCanvasContent.forEach((content, index) => {
    if (content.key === `${x}${y}`) {
      newCanvasContent.splice(index, 1);
    }
  });
  setCanvasContent(newCanvasContent);
  forceUpdate();
};
```

Figura 16. Código da remoção de semiose

6. Discussão

As principais mudanças entre as versões da ferramenta estão na quantidade de linhas de código. A versão anterior apresentava códigos muito acoplados e densos, que se provavam cansativos de compreender quando necessário efetuar até mesmo a mínima das alterações de alguma funcionalidade do sistema.

A Figura 17 exemplifica a quantidade de linhas presentes em um dos arquivos do EdiMM. Este arquivo era responsável por identificar e executar a semiose selecionada pelo usuário quando houvesse a requisição de adição do componente na tela.

A Figura 18 demonstra a grande redução de linhas de código para executar a inclusão de semioses na tela, uma função que é executada toda vez que o usuário seleciona qual componente deseja adicionar e clica na tela branca. Este código está presente no arquivo App.js, que é a centralização de todo o conteúdo do sistema, sendo o arquivo com maior quantidade de linhas de código, contendo a organização de cada uma das barras (superior e lateral esquerda) e a organização das semioses.

Outro ponto de extrema relevância é o baixo acoplamento, resultado da utilização do ReactJs, onde cada componente se responsabiliza em lidar somente com seu próprio contexto e suas próprias funções, diferente da versão antiga da ferramenta EdiMM, na qual vários arquivos executam diferentes funções.

Figura 17. Total de 3446 linhas de código no sistema anterior

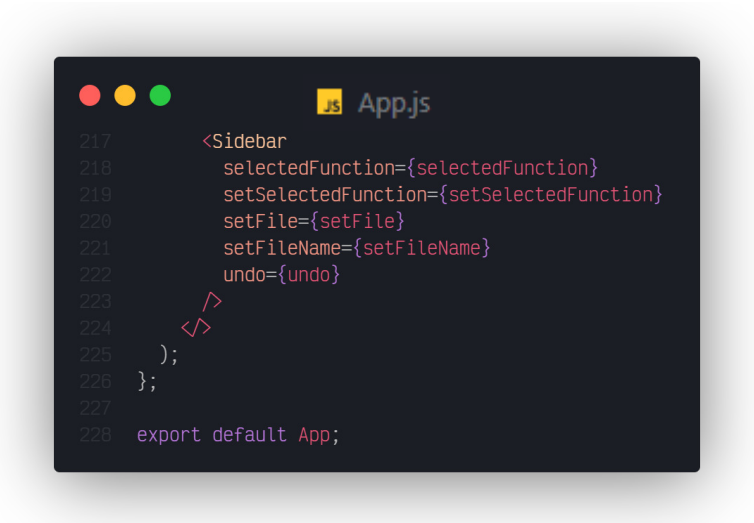


Figura 18. Total de 229 linhas de código no sistema atual

O código utilizado na Figura 19 demonstra como são renderizados todos os componentes do sistema.

App.js

```
return (
    <>
      <AppHeader
        brushRadius={brushRadius}
        setBrushRadius={setBrushRadius}
        setColor={setColor}
        color={color}
        fontSize={fontSize}
        font={font}
        setFont={setFont}
        setFontSize={setFontSize}
        bold={bold}
        setBold={setBold}
        underlined={underlined}
        setUnderlined={setUnderlined}
        italic={italic}
        setItalic={setItalic}
        handlePrint={handlePrint}
      />
      <div onClick={() => handleClick()}>
        <FileCanvas>{canvasContent}</FileCanvas>
        <CanvasDraw
          brushColor={color}
          brushRadius={brushRadius}
          disabled={selectedFunction === "draw" ? false : true}
          lazyRadius={0}
          style={{
            postion: "fixed",
            width: "100vw",
            height: "100vh",
            top: "0",
            background: "#ffffffff",
          }}
          hideGrid
        />
      </div>
      <Sidebar
        selectedFunction={selectedFunction}
        setSelectedFunction={setSelectedFunction}
        setFile={setFile}
        setFileName={setFileName}
        undo={undo}
      />
    </>
  );
};

export default App;
```

Figura 19. Exemplo de código do arquivo App.js usando React

7. Conclusão

Antes da realização deste trabalho, o EdiMM possuía problemas de manutenibilidade e alto acoplamento. Motivado em redesenvolver a ferramenta para que possibilite uma melhor manutenção e entendimento do seu código, além de proporcionar um sistema escalonável, este trabalho foi realizado. Vários problemas no código foram encontrados devido ao defasamento das tecnologias utilizadas para a criação da primeira versão. Em seguida, foram aplicadas as alterações utilizando o framework ReactJs. Pode-se concluir que, após o redesenvolvimento, a separação de funcionalidades e componentes se tornou mais visível e acredita-se que será de melhor compreensão.

Para a realização deste trabalho foram necessários conhecimentos aprofundados nas disciplinas de Desenvolvimento Web, Desenvolvimento de Sistemas Web, Arquitetura de Software, Engenharia de Software e Linguagem de Programação, Interação Humano Computador e Inglês Técnico.

A maioria do código do EdiMM se encontra melhor organizado e mais coeso, porém ainda há a necessidade de implementar as funções importantes restantes da ferramenta sendo algumas delas:

- Persistência dos dados - possibilidade de salvar e carregar a seção atual, gerando um link que pode ser acessado por qualquer um em qualquer lugar;
- Alteração - funcionalidade que possibilita movimentar componentes através da tela;
- Personalização de grade - funcionalidade que permite ao usuário adicionar uma grade como plano de fundo no canvas principal;
- Sincronicidade - permitir que a mesma seção atualize toda vez que alguma alteração seja feita.

Acreditamos que, com as modificações realizadas neste trabalho, será mais fácil a adição dessas funcionalidades.

Referências

BERNSEN, N. O. Multimodality Theory. In: TZOVARAS, D. (Ed.), Multimodal User Interfaces: From signal to interaction. Berlim, Alemanha: Springer-Verlag Berlin Heidelberg, 2008. p. 5-28.

EXCALIDRAW. Excalidraw. Disponível em: < https://excalidraw.com/ > . Acessado em: novembro de 2020.

FREIRE, F. M. P.; ARANTES, F. L.; SILVA, A. C.; VASCON, L. E. L. Estudo de viabilidade de um Editor Multimodal: o que pensam os alunos?. In: XX Congreso Internacional de Informática Educativa (TISE 2015), 2015, Santiago, Chile. Proceedings of TISE – Nuevas Ideas en Informática Educativa, 2015. v. 11. p. 109-119.

GOOGLE. Google Jamboard. Disponível em: < https://jamboard.google.com/ > . Acessado em: novembro de 2020.

HUNT, P. Why did we build React? **ReactJs.org**, 2013. Disponível em: < https://ReactJs.org/blog/2013/06/05/why-react.html >. Acesso em: 10 novembro de 2020.

OLIVEIRA, D. Uma proposta de arquitetura para Single-Page Applications. 11 de dezembro de 2017. Disponível em: < https://www.cin.ufpe.br/~tg/2017-2/djo-tg.pdf >. 10 novembro de 2020.

SOMMERVILLE, I. **Engenharia de Software**.10. ed. São Paulo: Pearson Education do Brasil, 2018.

TUNG, K. Developing a frontend application using ReactJs and Redux. Disponível em: < https://www.theseus.fi/bitstream/handle/10024/150837/Tung_Khuat_1301747_Thesis.pdf?sequence=1&isAllowed=y >. Acesso em: 15 novembro de 2020.

Documento Digitalizado Público

Anexo I - artigo - TCC

Assunto: Anexo I - artigo - TCC
Assinado por: Andre Constantino
Tipo do Documento: Relatório Externo
Situação: Finalizado
Nível de Acesso: Público
Tipo do Conferência: Documento Digital

Documento assinado eletronicamente por:

- **Andre Constantino da Silva**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 04/04/2022 22:18:18.

Este documento foi armazenado no SUAP em 04/04/2022. Para comprovar sua integridade, faça a leitura do QRCode ao lado ou acesse https://suap.ifsp.edu.br/verificar-documento-externo/ e forneça os dados abaixo:

Código Verificador: 936373
Código de Autenticação: 4f6f290fa5