UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
SISTEMAS DE INFORMAÇÃO

Camila dos Reis

Ambiente de desenvolvimento integrado para personalização automatizada do framework Bulma mediante variáveis do Sass

Florianópolis
2023

Camila dos Reis

Ambiente de desenvolvimento integrado para personalização automatizada do framework Bulma mediante variáveis do Sass

Trabalho de Conclusão de Curso submetido ao curso de Sistemas de Informação do Centro Tecnológico da Universidade Federal de Santa Catarina como requisito parcial para a obtenção do título de Bacharela em Sistemas de Informação.
Orientador: Prof. Elder Rizzon Santos, Dr.

Florianópolis
2023

Reis, Camila dos
Ambiente de desenvolvimento integrado para personalização automatizada do framework Bulma mediante variáveis do Sass / Camila dos Reis ; orientador, Elder Rizzon Santos, 2023.
66 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Sistemas de Informação, Florianópolis, 2023.

Inclui referências.

1. Sistemas de Informação. 2. Front-end. 3. CSS. 4. Pré-processador. I. Rizzon Santos, Elder. II. Universidade Federal de Santa Catarina. Graduação em Sistemas de Informação. III. Título.

Camila dos Reis

Ambiente de desenvolvimento integrado para personalização automatizada do framework Bulma mediante variáveis do Sass

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do título de Bacharel e aprovado em sua forma final pelo Curso Sistemas de Informação.

Florianópolis, 11 de dezembro de 2023.

Coordenação do Curso

Banca examinadora

Prof. Prof. Elder Rizzon Santos, Dr.

Orientador

Prof. Ismael Seidel, Dr.

Instituição Universidade Federal de Santa Catarina

Prof. Maicon Rafael Zatelli, Dr.

Instituição Universidade Federal de Santa Catarina

Florianópolis, 2023.

Para a minha mamãe, que até hoje dedica a própria vida a mim.
E ao nosso querido professor Leandro José Komosinski,
quem idealizou esse projeto.

AGRADECIMENTOS

Em primeiro lugar, quero agradecer à minha mamãe, por fazer eu ser quem sou, por não desistir de mim, por insistir que eu tivesse uma educação de qualidade e estar comigo desde sempre.

Ao meu namorado Lucas, que eu amo tanto, e que me apoiou esse tempo todo, mesmo durante meus choros e quando eu desacreditava de mim.

À minha tão meiga e querida amiga mosao, que foi a pessoa que mais me motivou a escrever o TCC e me mimava sempre que eu estava triste.

À minha amiga Sora, por sempre ter me ajudado em todos os quesitos: acadêmicos, pessoais e profissionais.

Aos meus amigos Paulo e John, que tive o prazer de conhecer em um dos meus primeiros estágios e se tornaram amigos para a vida toda.

Aos meus queridos amigos e colegas de curso, Alan, Alexandre (famoso "*pele de foca"*) e Fernando, que conheci durante o ensino à distância e que formavam os melhores trabalhos em grupo.

Ao pessoal da minha equipe, que fez questão de assistir minha defesa e ao meu chefe, Francisco, que permitiu home office para eu "dar um gás" no meu TCC.

Ao meu psiquiatra, Dr. Luiz Guilherme Silva Filho, que me ajudou durante as crises de ansiedade que quase me fizeram desistir de fazer o TCC.

Ao meu orientador, Elder Rizzon Santos, não só por ter aceitado ser meu orientador num momento extremamente difícil, mas por toda a paciência em explicar o que eu deveria fazer, em me motivar a cada e-mail trocado e principalmente por ser tão paciente, mesmo às vezes eu sumindo por conta de sérias crises de ansiedade.

Ao professor Leandro José Komosinski, meu primeiro orientador e idealizador desse projeto. Mesmo que infelizmente não se encontre mais entre nós, ele será eterno em cada conhecimento compartilhado conosco.

Aos professores da minha banca avaliadora, Ismael Seidel e Maicon Rafael Zatelli, por terem aceitado participar dessa missão comigo.

Ao Cislaghi, por ter me auxiliado e tirado tantas dúvidas, bem como ter indicado o professor Elder para ser meu novo orientador.

E, por fim, mas não menos importante, à UFSC, por proporcionar um ensino de qualidade e possibilitar realizar um dos meus sonhos: me formar em uma universidade federal.

RESUMO

Com o avanço das tecnologias, o desenvolvimento de uma folha de estilo para uma página web se tornou mais complexo por conta das novas funcionalidades do CSS. Consequentemente, os desenvolvedores passaram a adotar o uso de pré-processadores, de forma que seja possível suprir a carência de operações lógicas do CSS. O presente trabalho tem como objetivo propor um modelo para automatizar a criação de um arquivo CSS através da adoção de pré-processamento. Para gerar este arquivo, o usuário utiliza uma página web que funciona como um ambiente de desenvolvimento integrado de forma que seja possível personalizar o framework Bulma, sem a necessidade de ter conhecimentos prévios sobre pré-processamento. Neste ambiente de desenvolvimento integrado o tamanho e tipo da fonte, cor de botões, sombreamento, entre outros, são customizados para no final gerar o arquivo CSS com o auxílio da linguagem de pré-processamento Sass. Cada CSS criado pode ser testado na própria página. Por fim, é realizada uma comparação entre a personalização do Bulma com e sem a aplicação desenvolvida, e a aplicação deste TCC se mostrou muito mais rápida e simples de utilizar.

Palavras-chave: Front-end; CSS; Pré-processador.

ABSTRACT

As technology starts to become more advanced, the web page stylesheet development has become more complex due to new CSS funcionalities, and because of that developers started to use preprocessors, filling the logic functions missing on CSS. The present academic work aims to develop a model to automate the CSS file creation through preprocessing. To generate this file, the user access an specific web page that works like an integrated development environment and can customize Bulma framework, with no need of previous knowledge in preprocessing. In this integrated development environment the font size or font type, the color bottons, the shades, among others, are customized in order to generate a CSS file with the help of Sass preprocessing language. Each CSS file generated can be tested in the web page. At the end of this paper, a comparison is made between the customization of Bulma with and with and without the application developed, which proved to be much faster and simpler to use.

Keywords: Front-end; CSS; Preprocessor.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AJAX	Asynchronous Javascript and XML
CSS	Cascading Style Sheets
DHTML	Dynamic HTML
DOM	Document Object Model
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
MRU	Menos Recentemente Usada
MVC	Modelo Visão Controle
NPM	Node Package Manager
RIA	Rich Internet Application
Sass	Syntactically Awesome Style Sheets
URL	Uniform Resource Locator
XML	Extensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

Em desenvolvimento web, a divisão entre *front-end* e *back-end* ocorre desde 2009, com base em nosso levantamento. No início dos anos 90, para o desenvolvimento de sites dinâmicos, os desenvolvedores web configuravam o servidor e programavam os sites na linguagem PHP (ou linguagens com o mesmo propósito), no qual produziria algum código em HTML (Fender; Young, 2015). Foi somente em meados de 2009 com o progresso do JavaScript que esta classificação passou a existir, principalmente por conta do surgimento do Node.js, que permitia o funcionamento do JavaScript não só em navegadores (Błaszyński, 2018).

Graças a essa ascensão de tecnologias web e equipamentos computacionais, foi possível criar páginas web mais complexas e com mais funcionalidades. Com isso, tornou-se necessário criar a classificação *back-end*, para desenvolvimento da parte do servidor, e *front-end*, para o lado do cliente. É na programação do *front-end* que se desenvolve como a página web se comporta para que o usuário possa interagir (Fender; Young, 2015).

A área de desenvolvimento web abrange uma série de elementos: a começar pelo HTML (HyperText Markup Language), ou “linguagem de marcação de hipertexto”. Um hipertexto é uma espécie de escrita na qual possui links que concedem ao usuário a possibilidade de acessar rapidamente entre um texto e outro. Um documento HTML pode ser estático, onde os elementos após serem carregados não mudam de forma (a não ser que a página web seja carregada novamente), e dinâmicos, nos quais os elementos da página sofrem alterações mesmo após serem renderizados, sem a necessidade de carregar o site novamente. Isto permite que seja viável a responsividade, animações (com o CSS, por exemplo), validação de formulários e etc, mediante o auxílio de uma linguagem de cliente, como o JavaScript (Pfaffenberger; Schafer; White; Karow, 2004).

De acordo com Brooks (2007), o JavaScript, ou JS, é uma linguagem de programação interpretada criada para atuar em conjunto com o HTML de forma que as páginas se tornam interativas, através de um interpretador no navegador do usuário.

O CSS (*Cascading Style Sheets*), ou folha de estilo, é um mecanismo que permite formatar os elementos de um documento HTML, onde é possível definir um

estilo para uma página web. Com o CSS é possível modificar tamanhos de margens, fontes, cores ou até mesmo tabelas, sem a necessidade de codificação rígida (Pfaffenberger; Schafer; White; Karow, 2004).

Uma maneira de simplificar a construção do CSS é por meio do emprego de um *framework*, que é um conjunto de classes com a finalidade de possibilitar a utilização de funções recorrentes em um determinado tipo de software, ou seja, existem *frameworks* destinados a persistir dados em um banco, construir gráficos, entre outros, além de impor a arquitetura de uma aplicação (Gamma; Helm; Johnson; Vlissides, 1994). O Bulma, por exemplo, é um modelo de framework com código aberto para CSS criado por Jeremy Thomas, com vários elementos de HTML já prontos, como botões, modais, abas, paginação, etc. e que funcionam de forma responsiva (Bulma, 2021).

Segundo Phabhu (2015), o avanço do CSS3 trouxe novas funcionalidades para o CSS, como transições e animações, tornando-o mais complexo. Para expandir a manutenibilidade do código e compensar a carência de operações lógicas como funções, variáveis e demais operações, os pré-processadores CSS foram criados. Um exemplo de linguagem de folha de estilo é o Sass (*Syntactically Awesome Style Sheets*), que concede o uso de operadores numéricos, lógicos, de igualdade, funções, entre outros, e ao ser compilada, se torna um código em CSS (Sass: Syntactically Awesome Style Sheets, 2013).

Esta monografia propõe facilitar o emprego de pré-processadores, mediante a automatização da criação de um arquivo CSS (especificamente do *framework* Bulma) através da personalização do Sass em uma página web, que atuará como um ambiente de desenvolvimento integrado, seguindo o padrão de projeto MVC (*Model-View-Controller*). Existem vários tipos de padrão de projeto e cada um dita a forma que um software será estruturado, de maneira que possa atender melhor às necessidades ou problemas conhecidos para aquele determinado tipo de software. O MVC é um tipo de padrão de projeto dividido em três classes: *Model* (Modelo), que trata dos dados da aplicação, como questões de armazenamento e criação de dados; *View* (Visão), na qual representa a interface do usuário e *Controller* (Controlador), que funciona como uma ponte entre *Model* e *View*, pois é a parte responsável por obter os dados recebidos pela Visão e repassá-los para o Modelo, e vice-versa (Gamma; Helm; Johnson; Vlissides, 1994). Além disso, neste ambiente de desenvolvimento será

possível aplicar o CSS gerado na própria página, para prova de conceito, de forma que seja possível visualizar o resultado, realizar o download do CSS e criar classes de CSS personalizadas, além das já existentes no Bulma.

1.1 OBJETIVOS

Nas seções abaixo estão descritos o objetivo geral e os objetivos específicos deste TCC.

1.1.1 Objetivo Geral

Propor e analisar um novo modelo para automatizar a criação de um arquivo CSS mediante adoção de pré-processamento.

1.1.2 Objetivos Específicos

- Analisar o estado da arte em padrões de projeto para desenvolvimento web e abordagens específicas para *front-end*.
- Utilizando o resultado da análise realizada, desenvolver um modelo para a automatização de CSS baseado em pré-processamento.
- Analisar o modelo mediante um ambiente de desenvolvimento integrado para obter a prova de conceito.

2 FUNDAMENTAÇÃO TEÓRICA

A programação *front-end* tem se tornado cada vez mais presente em variadas formas, seja na digitalização de organizações ou transformando softwares de desktop em aplicações web, como por exemplo o Google Docs ou Word Online. Isto é vantajoso uma vez que uma mesma aplicação pode ser executada em diferentes sistemas operacionais, já que os navegadores compreendem HTML, CSS e JavaScript (Antunes e Fonseca, 2021). Kaushal et al. (2022) ressaltam que por conta da diferença de resolução da tela entre os diversos dispositivos existentes, os web sites podem apresentar um *layout* diferente do esperado, e para contornar esse problema, as aplicações precisam ser responsivas, ou seja, devem se adaptar de acordo com o tamanho da tela em que está sendo executada, e para isso é necessário utilizar modelos de *layout* como “flexbox” ou “CSS grid”. As diferentes versões de navegadores também podem não compreender tais modelos de *layout*, e isso pode ser contornado com o uso de linguagens de pré-processamento, como o Sass ou o Less, evitando, assim, a incompatibilidade entre os navegadores (Mazinanian e Tsantalis, 2017).

Segundo Delčev e Drašković (2018), os websites inicialmente possuíam somente uma linguagem estática (HTML), e com o avanço das tecnologias, viu-se necessário utilizar uma linguagem que permitisse que sites se tornassem mais dinâmicos, e assim surgiu o JavaScript. E por conta da crescente migração de aplicações desktop para aplicações web, passou a agir não somente no lado cliente, mas também no lado servidor, através de *frameworks*, como o Backbone.js, por exemplo.

O JavaScript também pode ser executado fora do navegador graças ao Node.js: um sistema de tempo de execução de código aberto, que possui vários pacotes gerenciados pelo NPM (Node Package Manager), como afirmam Goswami et al. (2020). De acordo com Wittern, Suter e Rajagopalan (2016), estes pacotes formam um ecossistema de aplicações e bibliotecas que funcionam tanto para o lado servidor quanto para o lado cliente.

3 TRABALHOS RELACIONADOS

A fim de obter resultados que auxiliassem no entendimento e desenvolvimento deste projeto, foi realizada uma pesquisa focada no uso de tecnologias relacionadas ao *front-end*, focando principalmente em pré-processadores.

3.1 AN EMPIRICAL SURVEY ON THE USE OF CSS PREPROCESSORS

Inicialmente as linguagens de estilo foram criadas para auxiliar desenvolvedores que não possuíam muita experiência com linguagem de programação, fazendo então com que carecessem de operações lógicas e limitando o uso do CSS. Para suprir essa falta de operações lógicas, os pré-processadores foram inventados.

De forma concisa, os autores explicaram como funcionavam conceitos básicos de linguagens de pré-processamento, além de realizar o propósito do estudo: analisar o uso das funcionalidades de pré-processadores em cerca de 150 páginas da web.

Com relação às variáveis, que possuem um uso semelhante ao das variáveis em linguagens de programação, notou-se que 89,29% delas são variáveis globais, sendo que a maior parte (45,98%) é utilizada para modificar cores de elementos no HTML.

Já a respeito do aninhamento, foi observado que possui uma definição análoga à herança na programação, ou seja, é com o aninhamento em que uma classe específica pode herdar atributos de uma “classe pai”. Segundo Mazinanian e Tsantalis (2016), os desenvolvedores faziam uma média de profundidade de 2 níveis de classes de aninhamento, além de que 78,02% das classes possuíam alguma herança.

Devido ao desprovimento de funções em uma folha de estilo, os mixins foram concebidos. Mazinanian e Tsantalis (2016) afirmam que 63% dos mixins possuem 1 ou nenhum parâmetro, além de serem reutilizados duas ou mais vezes.

Por fim, o último conceito elucidado é o de extensão, que também opera de forma similar à extensão em programação (conhecido como “extends”), onde ao utilizá-lo, há um agrupamento de classes que compartilham as mesmas características.

Os autores notaram que na maioria dos casos os desenvolvedores utilizam todas as funcionalidades de pré-processadores. Os dados obtidos podem ser utilizados para um aprendizado para escrever um melhor código de pré-processamento, com menos ambiguidade e muito mais conciso além de que o conhecimento obtido motiva o uso de pré-processadores, dadas todas as vantagens de reuso de código.

3.2 A SURVEY ON CSS PREPROCESSORS

Queirós (2017) inicia seu artigo de forma semelhante ao artigo apresentado na seção anterior deste trabalho, relatando a respeito do motivo pelo qual a folha de estilo e a linguagem de pré-processamento foram criadas. Neste artigo o autor realizou uma pesquisa a respeito de diferentes pré-processadores e como cada um se comportava, entre eles o Less, Sass e Stylus, e os escolheu com base na popularidade e tamanho da documentação, e as variáveis estudadas nesta pesquisa foram o nível de maturidade, cobertura e performance de cada uma destas linguagens.

Sobre a maturidade, foram analisadas as linguagens com mais impacto e popularidade na plataforma de repositórios GitHub, de acordo com a quantidade de *commits* (envio do estado atual do código para o servidor, para controle de versionamento), contribuições, *forks* (uma cópia local do repositório, de forma que o usuário consegue fazer alterações sem modificar a versão original) e novas versões das linguagens. Queirós (2017) utilizou a ferramenta Google Trends para verificar a quantidade de vezes que o nome das linguagens foi mencionado em pesquisas na internet com a atividade destas linguagens no GitHub. Como resultado verificou que Sass e Less possuem uma atividade muito maior que a Stylus, além de que o crescimento do Sass ultrapassou o Less, e mesmo que o Sass fosse uma linguagem mais antiga, o Less era muito mais popular.

Quanto à cobertura, Queirós (2017) fez uma análise entre os recursos disponíveis das linguagens de estilo, tais como mixins, variáveis ou *loops*. Todas as linguagens permitem o uso de variáveis, porém somente no Less não é possível mudar o valor de uma variável após a declaração da mesma, além de ser a única linguagem a possibilitar a declaração de uma variável após o uso, também conhecido

como variável *lazy*. A linguagem Stylus é a única que permite o atributo “*lookup*”, onde é possível obter o valor de um atributo previamente definido através de uma variável. Por fim, em todas as linguagens há interpolação (quando uma série de regras de estilo é atribuída a uma variável). E em todas as linguagens há o uso de mixins.

E para verificar a performance, o autor observou o tempo de compilação de cada linguagem em arquivos de tamanhos distintos, com um computador de sistema operacional Windows 10, processador Intel Core i7-6700K de 4GHz, memória RAM de 16GB e SSD e instalou o Node.js v6.10.2. Em seguida, para comparar as operações de pré-processamento, foi criado um arquivo JavaScript para os testes. Este arquivo geraria 3 novos arquivos de CSS nos tamanhos de 10KB, 100KB e 1000KB, para então verificar o tempo levado de cada compilação. Os resultados mostraram que o Sass possuiu o melhor desempenho entre as três linguagens, com Less em segundo lugar e Stylus em terceiro.

Sendo assim, ainda que o Sass tenha obtido melhores resultados nos três quesitos analisados (maturidade, cobertura e performance), tanto o Less quando o Stylus mostraram ter bons resultados e cada um possui suas peculiaridades. Não é possível dizer exatamente qual linguagem é melhor que a outra de forma geral, pois isto é uma opinião muito pessoal de cada desenvolvedor, que deve escolher uma linguagem de pré-processamento que seja melhor adequada às suas necessidades e que o desenvolvedor se sinta mais confortável em utilizar.

3.3 ARQUITETURA ORIENTADA A COMPONENTES PARA UMA WEB RESPONSIVA

No início da internet, as páginas web eram estáticas e foram desenvolvidas apenas para navegadores de computadores. Atualmente, celulares, televisores e tablets conseguem executar sites nativamente. E com a intenção de criar páginas que funcionassem de forma perfeita nos mais diversos dispositivos, Monteiro (2015) propõe criar uma arquitetura *front-end* que seja responsiva, utilizando os conceitos de *Web Components*, *Offline-first*, *Responsive Web Design* e *Single-Page Applications*, por serem conceitos que surgiram nos últimos anos e para que as páginas web que utilizarem essa arquitetura proposta possam se comportar como aplicações nativas

dos dispositivos em que irão funcionar, aumentando assim a compatibilidade de uma aplicação com os mais diversos tipos de computadores.

Os *Web Components* são componentes customizáveis de elementos do HTML. Assim como o HTML utiliza de forma nativa a *tag* “<img>” para a inserção de imagens, ou a *tag* “<form>” para a criação de formulários, por exemplo, com os *Web Components* é possível fazer a criação de uma nova *tag* que possa resolver um problema específico.

Já o *Offline-first* é um conceito que permite o uso de aplicações de modo offline, pois permite que haja um banco de dados do lado cliente, e com isso a aplicação funcionaria normalmente mesmo sem sinal de internet.

Para que as páginas tivessem um design responsivo em quaisquer dispositivos com telas de diferentes tamanhos, Monteiro (2015) utilizou o *Responsive Web Design*. Assim, imagens, fontes e outros elementos redimensionariam para que coubessem na tela; *layouts* adaptariam-se a diferentes ecrãs ou as páginas teriam sua estrutura simplificada em aparelhos móveis.

Por fim, a abordagem *Single-Page Application* faz o uso de uma única página web em que o usuário interage. Estas páginas únicas possuem mais funcionalidades e são mais complexas. Um exemplo amplamente conhecido de uma *Single-Page Application* é o Gmail.

Após unir todos estes conceitos, Monteiro (2015) criou uma arquitetura para páginas web que fosse responsiva e orientada a componentes. Basicamente o autor desenvolveu uma única página web composta por *Custom Elements*, com interface dinâmica e *layouts* responsivos, utilizando o *framework* gratuito e de código aberto, que auxilia no desenvolvimento web e móvel e possui uma arquitetura facilitadora para o desenvolvedor, pois assim a parte de programação *back-end* perde a necessidade de ser implementada (NoBackend) e por permitir o armazenamento de dados do lado cliente, fazendo assim com que a aplicação funcione de forma offline (*Offline-first*).

Para a implementação da prova de conceito, o autor desenvolveu uma aplicação web para receber pedidos de encomendas a restaurantes, bem como o acompanhamento do status do pedido em tempo real, chamada “Nomnow”, para iOS e Web.

Por fim, Monteiro (2015) compara o código desenvolvido para iOS e o código para uma página web. Enquanto a versão para iOS possuía 7203 linhas (somando os

códigos de Objective C, XIB, CSS e C/C++ Header), 1135 comentários, 3126 linhas em branco e 167 arquivos, a aplicação web possuía 7261 linhas (contabilizadas por número de linhas do código JavaScript mais HTML e CSS), 811 comentários, 569 linhas em branco e 109 arquivos. A contabilização destes dados só foi possível graças à ferramenta cloc (*Count Lines of Code*). O cloc é um arquivo único e executável que consegue contar linhas, comentários e linhas em branco em diversos tipos de linguagem de programação e pode ser utilizado em uma série de sistemas operacionais.

Foram feitos testes relacionados à responsividade em diversos dispositivos, que utilizam tanto mouse e teclado, quanto dispositivos que possuem ecrã tátil, como o Motorola Moto G 2014, Nokia Lumia 535, Apple iPhone 6, computador pessoal com resolução de 1920x1080, entre outros; em todos eles houve responsividade.

Segundo Monteiro (2015), houve sincronização de dados em tempo real e a navegação era fluída. Houve uma diminuição da performance em navegadores que não possuíam suporte nativo para *Web Components*, e para contornar essa situação "*polyfills"* foram utilizados para emular o suporte à navegação. Em relação à navegação offline, esta só ocorreu em versões recentes dos navegadores Google Chrome e Chrome for Android. E ainda assim, tornou a navegação mais rápida por não precisar acessar dados com o auxílio do *back-end*.

3.4 MVC WEB DESIGN PATTERNS AND RICH INTERNET APPLICATIONS

A evolução da internet causou um impacto nos mais diversos setores de nossa sociedade, e com isso as necessidades dos usuários foram crescendo em conjunto com esta evolução, e para supri-las, os desenvolvedores estão fazendo o uso das Aplicações de Internet Rica (da sigla inglesa RIA: *Rich Internet Application*), que são aplicações web com funcionalidades tão complexas quanto de programas desktop, tais como "arrastar e soltar" (em inglês, *drag and drop*). Com este artigo, os autores realizam uma discussão a respeito do propósito da arquitetura Modelo-Visão-Controlador (MVC) através de uma aplicação web e afirmam que este padrão pode fazer o uso das RIAs e diminuir o tempo e custo de processos de engenharia.

As vantagens da RIA não se dão apenas sobre os três principais aspectos da web (dados, apresentação e capacidade de comunicação), e sim, de um refinamento

entre cliente e servidor, diminuindo então a quantidade de atualização das páginas, requisições, ou filtragem de dados, por exemplo.

O padrão de projeto MVC passou a ser utilizado pela primeira vez através da linguagem de programação Smalltalk-80. É um padrão amplamente utilizado por manter a qualidade da aplicação e diminuir o tempo e custo de processos de engenharia, e é divido em três partes: Modelo, referindo-se à parte da persistência dos dados; Visão, na qual o usuário pode ver e interagir com a aplicação, e Controlador, a parte lógica do programa que manipula os dados entre a camada de Modelo e Visão.

Os autores focam nos seguintes modelos MVC: MVC do lado servidor, MVC híbrido com lado cliente e lado servidor e RIA MVC.

O MVC do lado servidor é baseado numa aplicação MVC desktop, onde o servidor realiza todas as operações para o cliente, e o navegador mostra toda a parte da visão. Este modelo de MVC é vantajoso quando se refere a aplicações mais simples com poucos recursos, como em navegadores de aparelhos celulares sem o uso de uma linguagem de programação do lado cliente, como o JavaScript. Por conta disso, é um modelo bastante simples onde só há um modelo, uma visão e um controlador. Já a desvantagem seria a necessidade de atualização constante da página para o controlador modificar a visão e aumento da utilização de banda.

Já o MVC híbrido com lado cliente e lado servidor visa resolver os problemas encontrados no MVC puramente no lado servidor, por conta da interação do usuário e de forma a diminuir todo o trabalho que antes era puramente feito pelo servidor, passando parte das manipulações de dados para as linguagens de lado cliente, como o JavaScript. Uma desvantagem é que a execução de linguagens no lado cliente possuem dependência do sistema do cliente e com isso pode tornar as páginas web mais lentas e pesadas. Este padrão de projeto também é conhecido como DHTML (ou HTML dinâmico). Algumas das desvantagens seriam a necessidade de ter parte do modelo no lado cliente (o que pode ser complicado, por conta da falta de compatibilidade de algumas funções com os navegadores, dispositivos e sistemas operacionais) e no lado servidor e acesso aos dados do modelo de forma assíncrona do servidor sem que haja uma atualização do site. Neste modelo também é possível utilizar AJAX, uma junção de HTML, JavaScript e obtenção de dados com XML assíncrono.

O último modelo a ser analisado é o RIA MVC, e segundo Morales-Chaparro et al. (2007), capaz de reduzir os problemas encontrados no MVC de lado servidor exclusivo e no híbrido. Com o RIA MVC, praticamente toda a aplicação está localizada no lado cliente. Uma desvantagem que ainda permanece é a necessidade de adaptar certas partes do código para que haja compatibilidade com os dispositivos e navegadores, como ocorre na forma híbrida. No formato unicamente lado servidor havia a necessidade de atualizar a página sempre que fosse preciso enviar um dado para o servidor, o que passa a não ocorrer mais no RIA MVC, bem como a redução do uso de banda de internet. Já em comparação ao modelo híbrido, o RIA MVC permite que haja obtenção de dados de forma assíncrona com o Modelo do lado servidor e aumenta a interação entre estas camadas, uma vez que parte do Modelo também se encontra no lado cliente.

Os autores citam que muitos desenvolvedores preferem deixar toda a aplicação no lado cliente, porém notaram que o ideal é distribuir as camadas do MVC tanto no lado cliente quanto no lado servidor: o Modelo passa a ser dividido em duas subcamadas tanto no lado servidor, com persistência de dados voláteis e não voláteis, quanto no lado cliente, com dados recebidos do Modelo do lado servidor e uma camada sem persistência com dados voláteis. Esta divisão em camadas ocorre também com o Controlador, sendo um Controlador passivo com ações engatilhadas com o tempo e um Controlador ativo, engatilhado por ações do usuário. Com isso, a manutenção se torna menor e o reuso de código se intensifica.

Morales-Chaparro et al. (2007) concluem então que todos os usos de MVC são úteis, e os conceitos de Aplicações de Internet Rica poderiam ser utilizados em qualquer modelo de MVC, porém haverão melhores resultados dependendo do tipo de aplicação, uma vez que nem todas as aplicações necessitam de processamento de dados no lado cliente, e como mencionado, o RIA MVC é melhor aproveitado quando é aplicado de forma distribuída entre as camadas do MVC tanto no lado cliente quanto no lado servidor.

3.5 OUTROS TRABALHOS

3.5.1 Front-end Website Performance Optimization Based on Web Storage in Html5

Páginas web que demoram demasiado tempo para carregarem costumam fazer com que os visitantes desistam de acessá-la. Além disso, boa parte do carregamento de um site é feito pela parte de *front-end*, e com a *cache* vazia, o tempo de resposta do arquivo HTML é menor.

Os autores propuseram utilizar dois métodos para medir a performance de uma página web, seja com otimização baseada em *web storage*, onde alguns arquivos são gravados no *web storage*, de forma a reduzir requisições http, tamanho de download de arquivo e também por ser mais rápido que *caches* http, ou então com o algoritmo MRU (Menos Recentemente Usada), onde um objeto não é salvo quando o seu tamanho excede o limite estipulado; o algoritmo MRU é utilizado em buffers de disco, por exemplo.

Baseando-se no algoritmo MRU, os autores perceberam que problemas de exceção que são originados por limitação de armazenamento podem ser amenizados quando o website faz uma relação entre a taxa de leitura e o tamanho do item, fiscalizando o espaço livre da *web storage*, administrando o tamanho do item e substituindo-o com a ajuda do algoritmo MRU.

3.5.2 Comparison of the compilation speed of the SCSS and LESS preprocessors

Os autores realizam uma comparação no tempo de compilação dos códigos de pré-processadores SCSS e Less. O Less é uma linguagem mais recente e baseada em JavaScript, enquanto o SCSS é baseado em Ruby.

Para os testes, uma aplicação web foi criada pelos autores para atuar como um transpilador, possibilitando o usuário de escolher com um menu expansível entre uma linguagem e outra arquivos de 10KB, 50KB, 100KB e 200KB tanto em Less

quanto em SCSS, onde cada arquivo será executado 5 vezes e o resultado do tempo de compilação será a média do tempo de compilação.

Por fim, os resultados obtidos neste artigo mostraram que o Less foi mais rápido no tempo de compilação, tanto nos arquivos de 10KB quanto de 200KB. Em contrapartida, o tamanho do código CSS gerado pelo SCSS é menor, por ter uma sintaxe mais otimizada, e, portanto, o tamanho do arquivo gerado é inferior ao do arquivo Less.

3.5.3 Automated Analysis of CSS Rules to Support Style Maintenance

Sabendo-se da dificuldade em manter um bom código em CSS (conciso, organizado, sem nenhuma classe inutilizada) por conta de suas bastantes regras, os autores deste artigo resolveram criar algoritmos que relacionam o código de estilo com os nodos da árvore DOM (um padrão de modelagem de páginas web durante o tempo de execução) de sua respectiva página, a fim de evitar classes desnecessárias, declarações que são substituídas por outras declarações ou até mesmo classes indefinidas. Estes algoritmos seriam aplicados utilizando o software CILLA.

O passo inicial foi montar um algoritmo que fizesse a relação entre o CSS e o DOM, onde o algoritmo identifica todas as classes do CSS que foram e não foram utilizadas. Com isso, é possível seguir para o segundo algoritmo, no qual identifica uma classe como “efetiva” caso possua pelo menos um elemento do DOM que tenha seu valor modificado por esta classe. Por fim, no último algoritmo é identificado quando uma classe é indefinida, ou seja, caso ela exista no DOM, mas não exista no CSS, ou definida, que é a situação inversa.

De todas as páginas analisadas, os autores identificaram que 60% das classes e 52% dos atributos do CSS são inutilizados, o que são números bastante extensos e com estes algoritmos, seria possível ter um melhor código de folha de estilo, o que melhoraria a qualidade e a manutenibilidade de um arquivo CSS, e consequentemente, tornaria o carregamento de uma página web mais rápido, por exemplo.

3.5.4 CSS preprocessor - Sass eller Less

Existem uma série de linguagens de pré-processamento, e sabendo disso, o autor resolveu em sua dissertação de mestrado, realizar uma comparação entre as linguagens de pré-processamento Less e Sass, na intenção de escolher qual delas é a melhor linguagem.

Ambas as linguagens possuem bastantes semelhanças na sintaxe, mas o autor aponta algumas diferenças, como pode-se observar no Quadro 1 a seguir:

Quadro 1 – Diferenças entre Sass e Less levantadas pelo autor

Sass	Less
Feito em Ruby e processado no lado servidor.	É uma biblioteca do JavaScript (ou seja, é processada no lado cliente).
Possui operadores lógicos e de repetição (tais como if, then, else, for, while e each).	Possui mixins que funcionam sob certas condições impostas pelo desenvolvedor.
Utiliza “$” para declarar uma variável.	Utiliza “@” para declarar uma variável.

Fonte: adaptado de Laukkanen (2017)

Levando estes fatores em conta, o autor elege o Sass como a melhor linguagem de processamento, principalmente por possuir atributos que o Less não tem, além de que o Sass foi criado antes e o Less foi baseado nele. De qualquer forma, ambos são excelentes linguagens de pré-processamento, e conhecendo a sintaxe de um, seria simples migrar para a outra linguagem.

3.6 CRITÉRIO DE AVALIAÇÃO DOS TRABALHOS RELACIONADOS

Para melhor analisar alguns dos trabalhos relacionados, o Quadro 2 foi criado com alguns critérios de avaliação. Nas seções seguintes haverá a relação dos trabalhos com os critérios criados.

Quadro 2 – Critérios de avaliação dos trabalhos relacionados

Critérios de avaliação	
Formato de entrada	Como é inserido ou criado o código CSS no contexto do trabalho relacionado.
Facilidade de uso	**Fácil**, onde não é necessário um conhecimento prévio de linguagem de pré-processamento; **médio**, no qual é preciso ter uma base de linguagem de pré-processamento; **difícil**, quando pré-processamento precisa ser dominado pelo usuário.
Manutenção/Atualizações	O modo em que o código CSS ou arquivo de linguagem de pré-processamento é modificado após ficar pronto.
Qualidade do CSS	**Baixa**, em que o código não possui um ordenamento correto de classes, mal indentado, com muitas classes repetidas ou inutilizadas; **média**, no qual o código é organizado e bem indentado, porém possui algumas classes repetidas ou inutilizadas; **alta**, código organizado, enxuto e bem indentado, como o do CSS gerado pelo Bulma.
Classes personalizáveis	De que forma as classes geradas pelo usuário são criadas.
Responsividade	Se o código gerado é responsivo em diferentes plataformas e proporções de tela.
Tamanho do arquivo gerado	Verificar o tamanho do arquivo CSS obtido com o uso da linguagem de pré-processamento.

Fonte: elaborado pela autora

3.6.1 Avaliação de “Comparison of the compilation speed of the SCSS and LESS preprocessors”

a) Formato de entrada: arquivos no formato Less e SCSS, nos tamanhos 10KB, 50KB, 100KB, 200KB;
b) Facilidade do uso: difícil, uma vez que para as análises realizadas foi necessário um conhecimento a respeito de linguagens de pré-processamento;
c) Manutenção/Atualizações: os códigos utilizados já estão prontos, eles apenas são compilados.
d) Qualidade do CSS: dado não presente no trabalho;
e) Classes personalizáveis: dado não presente no trabalho;
f) Responsividade: dado não presente no trabalho;
g) Tamanho do arquivo gerado: no caso do SCSS, os arquivos nos tamanhos de 10KB, 50KB, 100KB, 200KB se transformaram respectivamente em arquivos CSS nos tamanhos de 9KB, 45KB, 89KB e 182KB, enquanto para os arquivos Less foram arquivos de 10KB, 48KB, 97KB e 193KB.

3.6.2 Avaliação de “Automated Analysis of CSS Rules to Support Style Maintenance”

a) Formato de entrada: um arquivo CSS próprio dos autores, utilizado como exemplo e mais 15 arquivos CSS de diversos web sites;
b) Facilidade do uso: não há menção de uso de linguagem de pré-processamento;
c) Manutenção/Atualizações: no caso do código exemplo, totalmente manual, dado que o código CSS utilizado foi criado pelos autores; quanto aos dos web sites utilizados como exemplo, não é possível ter certeza;
d) Qualidade do CSS: para o exemplo utilizado é baixa, em relação aos exemplos de código HTML exibidos no artigo, mas pode-se dizer que é proposital, pois a proposta é analisar justamente a qualidade do código; dos web sites utilizados para análise, de acordo com alguns gráficos pode-se dizer que é de média a baixa, uma vez que a variação de classes inutilizadas em alguns web sites vai de pouco mais de 40% até quase 100%;
e) Classes personalizáveis: quanto ao exemplo dos autores é possível adicionar, porém manualmente; quanto aos dos web sites utilizados para análise, não é possível afirmar;
f) Responsividade: não há menção de elementos responsivos no código exemplo;
g) Tamanho do arquivo gerado: não é citado o tamanho do arquivo gerado, porém é reportado que um código CSS bem escrito pode gerar uma redução de tamanho de até 58%, simplesmente por eliminar propriedades inutilizadas.

3.6.3 Avaliação de “CSS preprocessor - Sass eller Less”

a) Formato de entrada: o código utilizado neste trabalho é inserido manualmente pelo autor em uma IDE;
b) Facilidade do uso: difícil, uma vez que para a análise entre uma linguagem de pré-processamento e outra, o autor precisou ter um conhecimento a respeito destas linguagens;

c) Manutenção/Atualizações: totalmente manual;
d) Qualidade do CSS: média, pois o código não seguiu um padrão de espaçamento de linha entre uma classe e outra;
e) Casses personalizáveis: sim, há classes personalizadas, porém todas feitas manualmente;
f) Responsividade: não há elementos de responsividade nos códigos utilizados neste trabalho;
g) Tamanho do arquivo gerado: nada é mencionado.

3.7 TABELA COMPARATIVA ENTRE OS TRABALHOS RELACIONADOS ANALISADOS

Com base nos trabalhos relacionados analisados, a Tabela 1 foi criada para exibir de forma resumida os resultados da avaliação.

Tabela 1 – Comparação entre os trabalhos relacionados analisados

	Comparison of the compilation speed of the SCSS and LESS preprocessors	Automated Analysis of CSS Rules to Support Style Maintenance	CSS preprocessor - Sass eller Less
Formato de entrada	Arquivo Less	Arquivo CSS	Manual
Facilidade de uso	Difícil	N/A	Difícil
Manutenção/Atualizações	Códigos prontos	Manual	Manual
Qualidade do CSS	N/A	Baixa	Média
Classes Personalizáveis	N/A	Manual	Manual
Responsividade	N/A	N/A	N/A
Tamanho do arquivo gerado	9KB a 193KB	N/A	N/A

Fonte: elaborado pela autora

3.8 DISCUSSÃO

Dentre os trabalhos relacionados analisados, destaca-se principalmente a necessidade de manter um código CSS de qualidade: conciso, sem classes ou

atributos inutilizados e bem indentado, uma vez que isso não somente auxilia o desenvolvedor na manutenção e leitura do código, como também torna uma página web mais rápida. Sendo assim, estes itens serão levados em conta no desenvolvimento deste projeto.

Com os diferentes dispositivos e navegadores presentes no mercado, uma página web precisa ser totalmente responsiva para que funcione perfeitamente, o que vai de encontro ao *framework* Bulma que é igualmente responsivo.

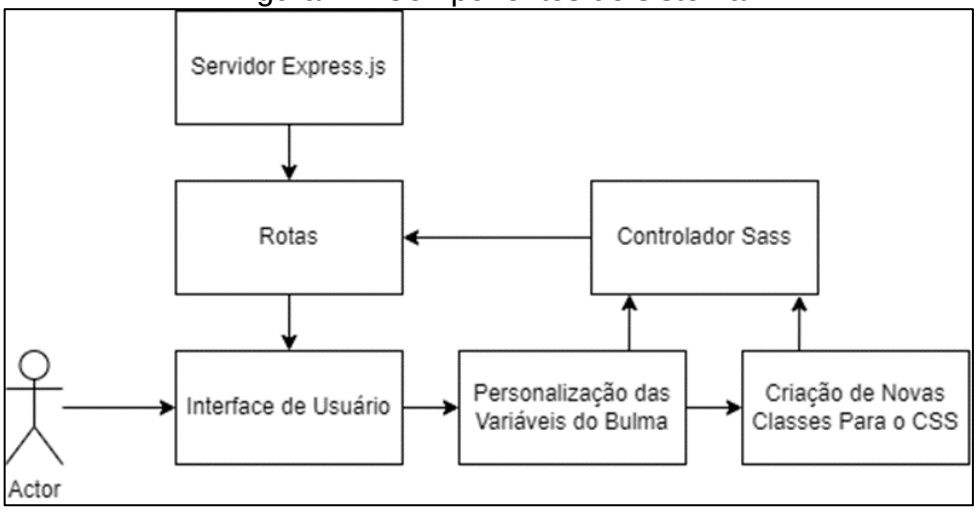
É necessário salientar que neste projeto o CSS gerado com a adição de novas classes customizáveis pelo usuário pode acarretar em classes e atributos que não serão utilizados por nenhum elemento do HTML, o que diminuiria a performance do website que o utilizasse. Além disso, pela liberdade do usuário de criar classes personalizáveis, é possível que estas não tenham nenhum atributo responsivo, o que poderia causar um mau funcionamento da página web em alguns dispositivos.

4 DESENVOLVIMENTO

A solução proposta é composta de seis componentes, como é possível observar na Figura 1, visando um modelo para automatizar a criação de um arquivo CSS mediante adoção de pré-processamento, que serão detalhados em subseções a seguir. Estes componentes estão organizados para funcionar de acordo com o modelo MVC, e rodará em um servidor Node.js, além de fazer a utilização de JavaScript puro, HTML, CSS e Sass.

A “Interface de Usuário” é a visão do sistema, onde o usuário pode interagir para criar o arquivo CSS personalizado. Este componente se comunica com “Personalização das Variáveis do Bulma”, que é a parte que recebe os dados inseridos no formulário, assim como o componente “Criação de Novas Classes Para o CSS” e ambos se comunicam com o arquivo “scripts.js”, onde possuem seus dados filtrados e tratados para serem enviados ao “Controlador Sass”, através de uma requisição que é tratada em “Rotas”, tudo isso rodando em um servidor, no “Servidor Express”.

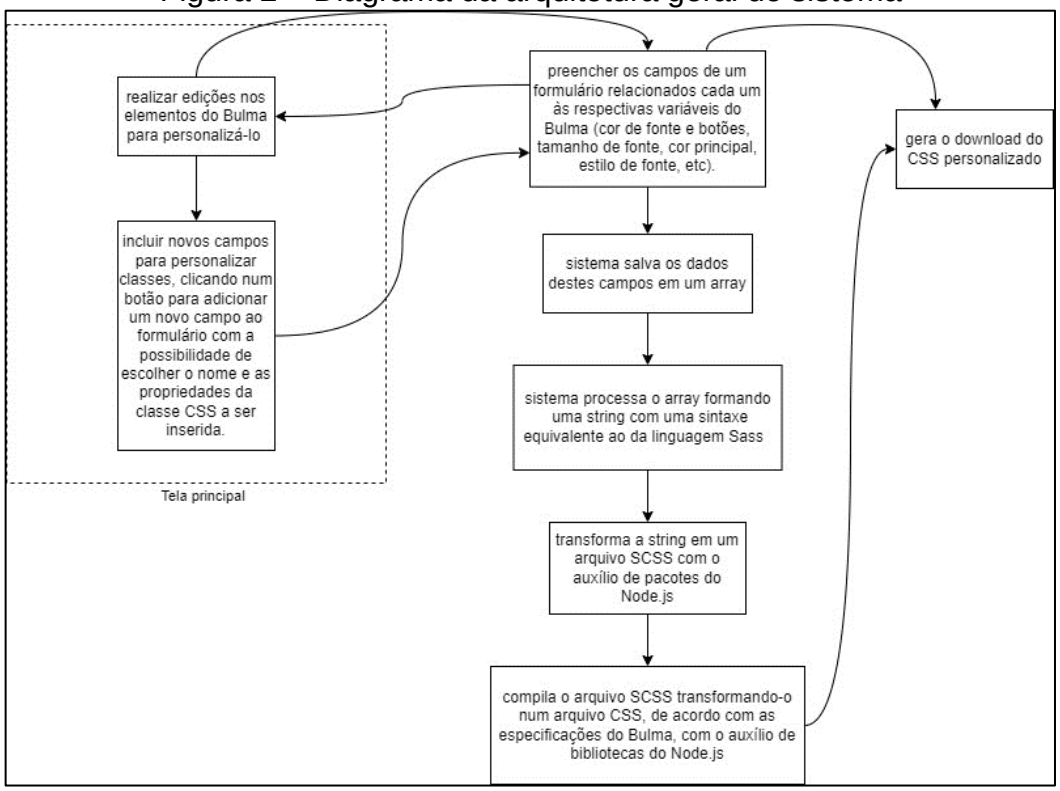
Figura 1 – Componentes do sistema



Fonte: elaborado pela autora

Para detalhar o fluxo do funcionamento do sistema deste projeto, a Figura 2 contendo um diagrama será utilizada e em seguida cada um de seus blocos serão elucidados.

Figura 2 – Diagrama da arquitetura geral do sistema



Fonte: elaborado pela autora

Inicialmente tem-se a tela principal do sistema, que é uma página web que contém abas para levar o usuário às áreas de criação de um novo arquivo CSS do Bulma e adicionar classes personalizadas ao CSS final.

No próximo bloco à direita, é possível escolher as variáveis do Bulma a serem customizadas (o que normalmente é possível fazer com o uso de linguagem de pré-processamento), tais como as cores e os tamanhos de botões, links e fontes. De forma diferencial ao que o Bulma normalmente permite, a aplicação admitirá que o usuário crie novas classes com atributos para a folha de estilo.

A customização dos elementos se dá por um formulário, em que cada campo é equivalente a uma das variáveis do Bulma. Para a criação de novas classes, basta clicar no botão “+” para que em tempo real a página adicione um novo campo ao formulário.

Ao finalizar a personalização, o usuário clica no botão “Compilar Sass”. Neste instante a primeira coisa que o sistema faz é unir cada dado dos campos do formulário em um *array* e monta um texto com a sintaxe da linguagem de pré-processamento Sass. Em seguida, as funções das bibliotecas do Node.js irão transformar o texto

criado no passo anterior em um arquivo SCSS e posteriormente compilam este SCSS em um arquivo CSS contendo as características do Bulma e possivelmente com as novas classes criadas. Por fim, é possível baixar o arquivo CSS, testá-lo em tempo real na própria aplicação ou continuar editando-o.

4.1 PERSONALIZAÇÃO DAS VARIÁVEIS DO BULMA

Nesta aba, o usuário realiza a customização das variáveis do Bulma através de um formulário que possui campos de entrada de dados, tais como menus expansíveis e seletores de cor. O *framework* Bulma possui variáveis do Sass que permitem uma personalização caso o desenvolvedor não queira as cores e tamanhos de elementos ou famílias e tamanhos de fontes padronizadas pelo *framework*.

Assim que o usuário modificar os campos conforme desejado, ao clicar no botão “Compilar Sass”, habilita-se a caixa de seleção “Testar CSS” para em tempo real verificar o resultado da customização do Bulma e o botão “Baixar CSS” que permite baixar o arquivo CSS já compilado.

Com o auxílio da linguagem JavaScript e manipulação de cada elemento DOM, todos os dados são filtrados e é possível montar um texto com a sintaxe do Sass para ser compilado posteriormente. A compilação do Sass é feita com o método “renderSync()” do pacote “sass” do Node.js.

O arquivo CSS baixado pelo usuário é gerado logo após a compilação do Sass, utilizando JavaScript puro, com instanciação de um objeto “Blob”. Este arquivo pode ser baixado após manipular o elemento DOM do botão “Baixar CSS”, que recebe um diretório contendo o CSS e a propriedade de download.

4.2 CRIAÇÃO DE NOVAS CLASSES PARA O CSS

Esta funcionalidade permite que novas classes CSS sejam adicionadas ao arquivo CSS compilado, e é totalmente opcional. Além disso, o usuário pode ou não posteriormente ter customizado alguma variável do Bulma.

Para criar a primeira classe personalizada, basta acessar a aba “Criar Classes” e em seguida clicar em “Clique aqui para começar!”, onde um evento do JavaScript será disparado e exibirá uma série de elementos de formulário para

adicionar algum atributo de classe CSS, como “display”, “color”, “font-size”, “width”, “flex-direction”, etc. Todos estes elementos são gerados de forma dinâmica, acessando os dados de um arquivo JSON contendo todas estas propriedades e seu determinado tipo (para mapear quando um elemento será um seletor de cor, menu expansivo com os respectivos possíveis dados de cada atributo), e cada modelo de elemento de formulário será gerado utilizando um módulo do JavaScript, contendo todos os modelos de formulário utilizados na solução proposta deste trabalho.

Cada um dos possíveis atributos possui logo abaixo de si uma caixa de seleção para o usuário selecioná-lo caso queira que o respectivo elemento esteja presente na classe criada. Toda a lógica para habilitar ou não um atributo é realizada com JavaScript puro, onde o evento é escutado e remove a pseudo-classe “disabled” deste elemento, permitindo então atribuir valores a este atributo.

Todas as classes personalizadas necessitam não apenas ter um nome, mas possuir um nome válido, caso o contrário, não é possível compilar o código Sass. Uma verificação de cada um dos nomes das classes é realizada, verificando se o nome da classe está vazio ou se possui algum caractere que não seja letras, números, hífen ou traço baixo (“_”), com o auxílio de uma expressão regular. Além disso, cada elemento de formulário com um nome inválido informa ao usuário através de um texto de suporte qual problema ocorreu para impedir a compilação. A aparência deste texto de suporte se dá pelo uso das classes “is-danger” e “help” do Bulma, onde definem, respectivamente, a cor e a aparência da fonte do texto.

Para adicionar mais de uma classe, o usuário deve clicar no botão com o ícone “+”, no qual chama o mesmo método escutado pelo evento do botão “Clique aqui para começar!”, para criar mais uma sequência de elementos de formulário contendo todos os atributos para personalizar uma classe do CSS. Todos estes elementos possuem um identificador único, para facilitar a manipulação do DOM. E para remover a classe personalizada, um botão com um ícone de lixeira sempre estará disponível, e ao ser clicado, remove a classe e os atributos atreladas a este botão, de acordo com o identificador.

4.3 SERVIDOR EXPRESS

Algumas funções de alguns pacotes do Node.js necessárias para o funcionamento da solução proposta por este trabalho só funcionam em um servidor Node.js (como por exemplo a função “renderSync()” do pacote “sass”), por isso o uso de um servidor se torna fundamental.

O servidor utiliza o pacote “express” do Node.js e cria uma instância de uma aplicação Express, e é com esta instância, junto de outros pacotes do Node.js que serão discorridos a seguir, que o servidor funcionará.

O pacote “path”, um módulo que auxilia a manipular diretórios de arquivos, criará a configuração das pastas da *view* (o “V” – visão – do modelo MVC), junto do pacote “ejs”, um mecanismo que permite a modelagem de um arquivo HTML com JavaScript.

Como a aplicação funciona basicamente analisando dados de formulários para gerar um arquivo CSS, é necessário utilizar o pacote “body-parser”, para examinar o corpo das requisições realizadas nestes formulários.

Por fim, é feita uma associação ao arquivo “routes.js” à rota raiz da aplicação, indicando como serão as rotas das requisições (por exemplo: “http://localhost:3000/compile-sass”), além de configurar o servidor para funcionar na porta 3000.

4.4 ROTAS

Todas as requisições da aplicação, por utilizar o Express, precisam ter suas rotas definidas, por isso, uma instância de um objeto de roteamento do Express é criada, do tipo “Router()”.

A primeira requisição é uma HTTP, do tipo “get”, que cria uma rota no diretório raiz da aplicação para renderizar um arquivo chamado “index”, que é a página inicial.

Uma requisição “post”, a “/compile-sass”, é configurada, passando como parâmetro a função “compileSass()”, do módulo “sassController.js”, que faz parte do controlador (o “C” do modelo MVC) da aplicação, para compilar o código Sass e transformar em um arquivo CSS do Bulma.

Por fim, é feita a exportação do objeto de roteamento criado, para que este possa ser utilizado por outros arquivos do projeto.

4.5 CONTROLADOR SASS

Este componente possui o elemento essencial do projeto, que é a função compiladora de um texto com a sintaxe do Sass em um arquivo CSS. A função “compileSass()” é criada e utiliza a função “renderSync()” do pacote “sass” para receber um texto já na sintaxe do Sass, contendo todos os elementos da visão, sejam das variáveis personalizadas do Bulma ou das variáveis criadas pelo usuário, pela requisição “/compile-sass”. Além disso, junto deste texto é realizada também a importação do Bulma, para gerar um CSS com as propriedades deste *framework*. Com o retorno de “renderSync()”, é possível criar um arquivo CSS chamado “test.css” (utilizado para testes do arquivo gerado) no diretório “../public/css”, com o auxílio dos pacotes “path” e “fs”.

De modo que a função “compileSass()” possa ser chamada em outros arquivos do projeto, a exportação do controlador é feita ao final do arquivo.

4.6 INTERFACE DE USUÁRIO

Toda a interface é feita utilizando a sintaxe de HTML, além de fazer a importação de arquivos CSS (tanto o padrão da aplicação quanto o gerado para testes) e ícones do Font Awesome, para auxiliar na usabilidade. A interface utiliza por padrão o CSS gerado pelo Bulma e um outro arquivo CSS criado para personalização de elementos específicos, como o seletor de cor, que foi criado manualmente e não utiliza o Bulma, pois este seletor é um botão com uma aparência personalizada (um círculo na cor desejada, em vez de ser um seletor padrão de cor do HTML).

Logo na página inicial da aplicação, o usuário pode escolher o que deseja fazer primeiro, se é personalizar o Bulma ou adicionar novas classes ao CSS compilado, clicando na aba correspondente. O usuário pode navegar entre uma aba ou outra, sem perder as alterações feitas, além de que as opções para compilar, baixar e testar o CSS estarão sempre disponíveis em ambas as abas.

4.7 PROVA DE CONCEITO

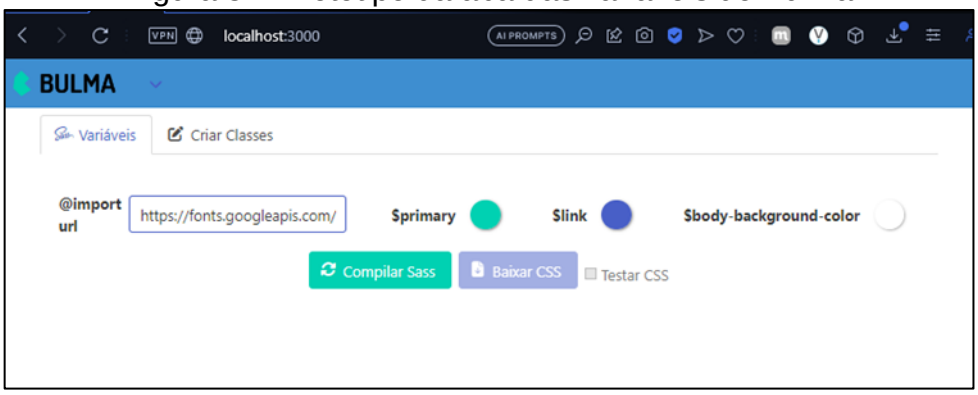
Durante a fase de prototipação da aplicação, foi desenvolvida a tela inicial, contendo duas abas: “Variáveis”, onde é possível modificar as variáveis do Bulma e “Criar Classes”, para gerar classes personalizadas.

Na primeira aba, como pode-se observar na Figura 3, possui 4 das 419 variáveis do Bulma. Seguindo a ordem da esquerda para a direita, o usuário poderá escolher uma fonte do Google Fonts e colar a URL no campo “@import url”, para realizar a importação da fonte e incorporá-la ao arquivo CSS compilado. O formato da URL seria algo semelhante a “https://fonts.googleapis.com/css?family=Handlee”, onde “Handlee” seria uma das fontes disponibilizadas no Google Fonts.

Os próximos três elementos são botões seletores de cor, estilizados com CSS para obterem esta aparência circular. Na Figura 4, a cor da variável “$primary” está sendo modificada.

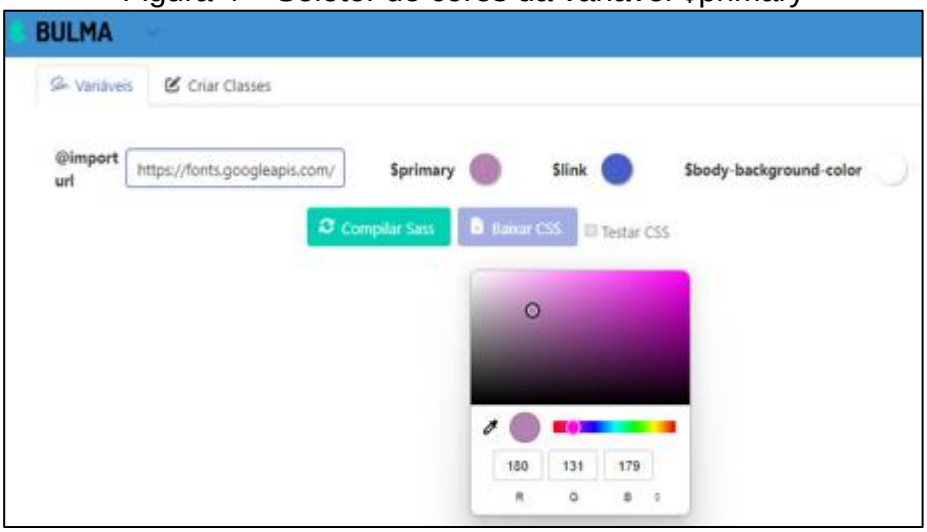
Quando o botão “Compilar Sass” é clicado, respectivamente “Baixar CSS” e “Testar CSS” passam a ser habilitados, permitindo então o download do arquivo CSS e também um teste deste arquivo recém compilado. Como o projeto também utiliza o Bulma para estilização, testar este CSS gerado irá alterar os elementos “$primary”, “$link”, “$body-background-color” e a fonte, como é possível verificar na Figura 5.

Figura 3 – Protótipo da aba das variáveis do Bulma



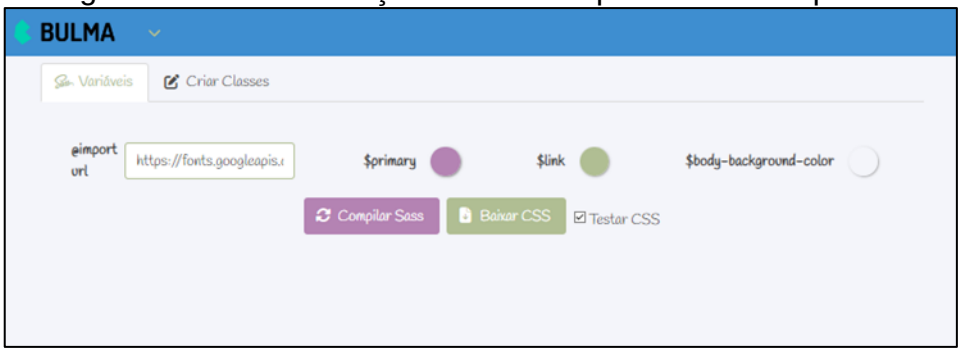
Fonte: elaborado pela autora

Figura 4 – Seletor de cores da variável $primary



Fonte: elaborado pela autora

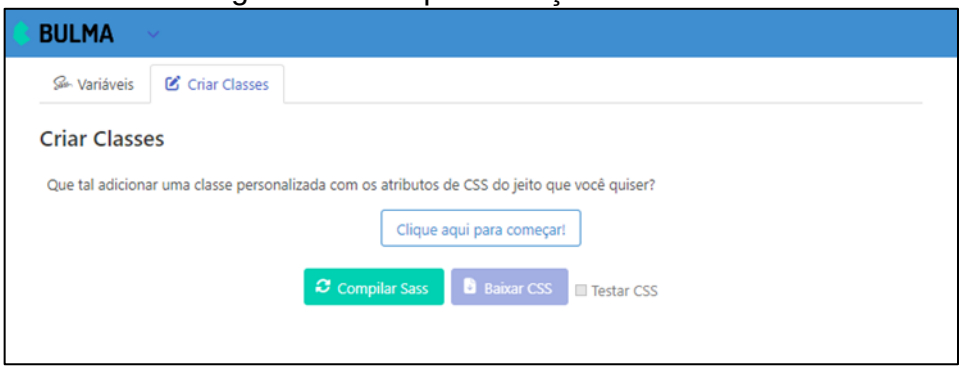
Figura 5 – Personalização do Bulma aplicada em tempo real



Fonte: elaborado pela autora

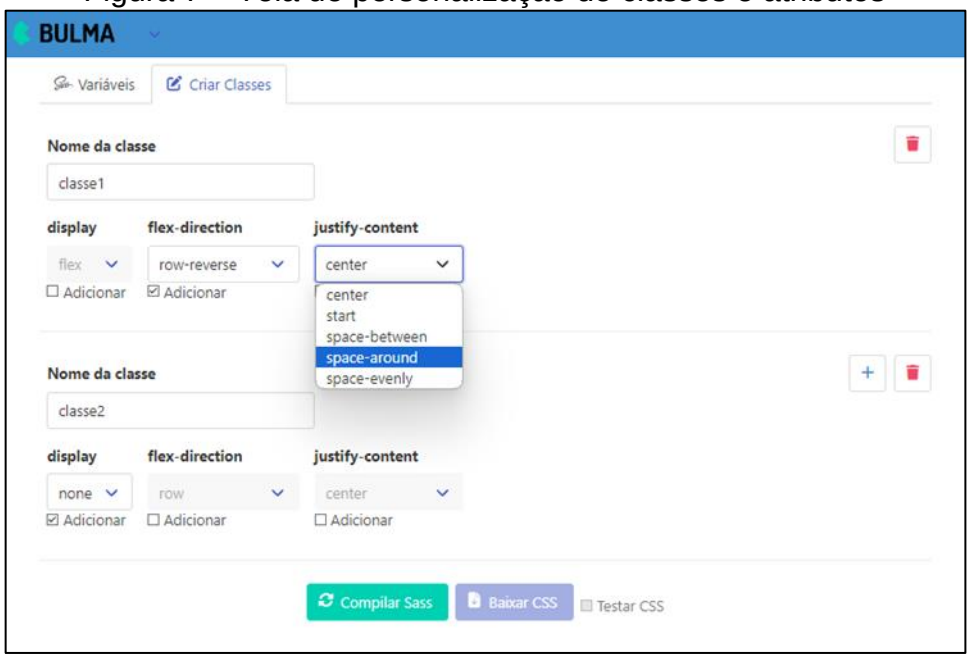
Na aba “Criar Classes”, o usuário tem a possibilidade de adicionar novas classes ao CSS compilado, com possibilidade de personalizar nome e atributos de classe.

Figura 6 – Aba para criação de classes



Fonte: elaborado pela autora

Figura 7 – Tela de personalização de classes e atributos



Fonte: elaborado pela autora

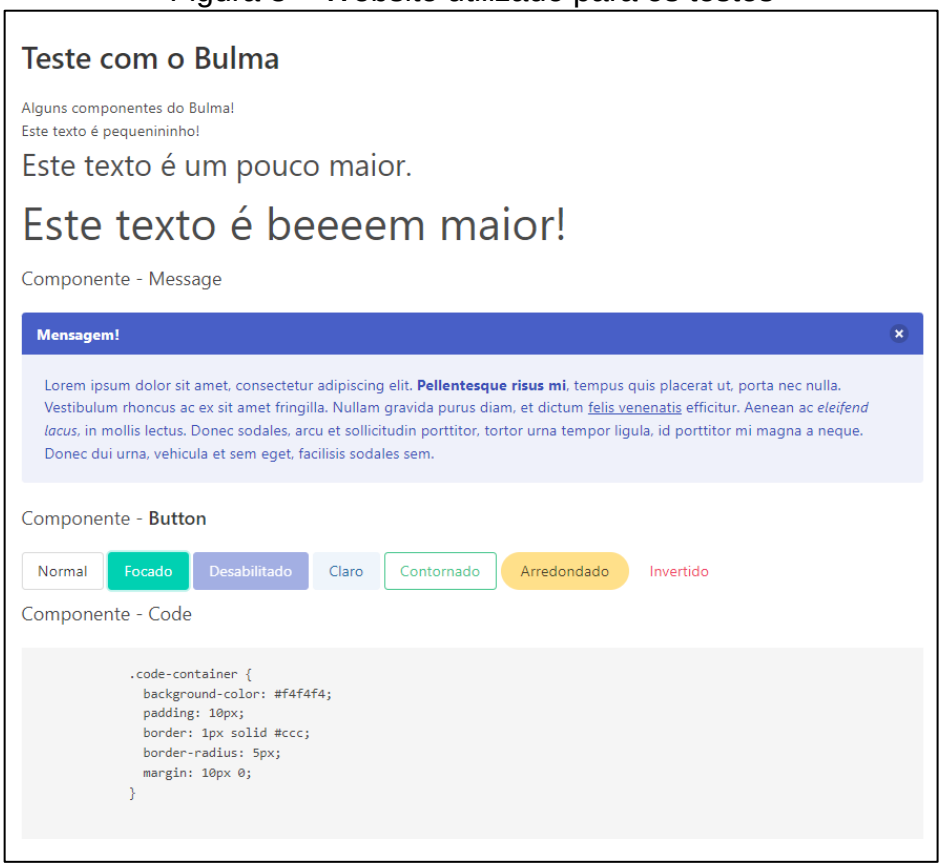
4.8 EXPERIMENTOS

Esta seção realiza dois experimentos para verificar as diferenças tanto do modo padrão de personalização do Bulma quanto com a solução proposta por este trabalho. Para verificar esta diferença, na primeira e segunda seção a seguir são explicados o passo a passo da instalação e execução da personalização das variáveis, para enfatizar todo o processo que é necessário normalmente, sem a aplicação desenvolvida.

Nas subseções seguintes, é documentada a comparação entre um processo e outro, com métricas de tempo de duração, quantidade de cliques de caracteres digitados para personalização do *framework* Bulma em um mesmo *site* e com as mesmas variáveis e mapa de calor da tela. Além disso, também é analisado o uso da aplicação e sem para criar novas classes para o CSS.

Sem customização alguma, o site possui a seguinte aparência (Figura 8):

Figura 8 – Website utilizado para os testes



Fonte: elaborado pela autora

As variáveis utilizadas nos testes serão as mesmas em ambos os testes, e segue a lista das variáveis no Quadro 3 a seguir:

Quadro 3 – Variáveis utilizadas nos testes
(continua)

Variáveis do Bulma	
Nome	**Tipo**
$primary	Cor
$info	Cor
$success	Cor
$warning	Cor
$danger	Cor
$background	Cor
$link	Cor
$strong-color	Cor
$body-color	Cor
$body-background-color	Cor
$pre-background	Cor
$pre	Cor
$custom-colors (azul)	Color
$family-primary	Fonte

Quadro 3 – Variáveis utilizadas nos testes (conclusão)

Variáveis do Bulma	
Nome	**Tipo**
$size-1	Tamanho
$size-3	Tamanho
$size-6	Tamanho
$pre-code-font-size	Tamanho
$code-padding	Tamanho

Fonte: elaborado pela autora

As classes personalizáveis são exibidas no Quadro 4, que também serão as mesmas classes e mesmos atributos em ambos os testes:

Quadro 4 – Classes personalizadas

Classes do CSS	
Nome da classe	**Atributos**
titulo	background-color outline-color letter-spacing opacity text-decoration outline-width outline-style
codigo	width text-align line-height border-style border-radius font-size

Fonte: elaborado pela autora

4.8.1 Instalação do Bulma

Há quatro formas de instalar o Bulma:

a) utilizar a importação do CSS através de uma rede de fornecimento de conteúdo, com uma URL, seja no código CSS (Figura 9) ou HTML (Figura 10);

Figura 9 – Importação do Bulma feita em um arquivo CSS

Fonte: elaborado pela autora

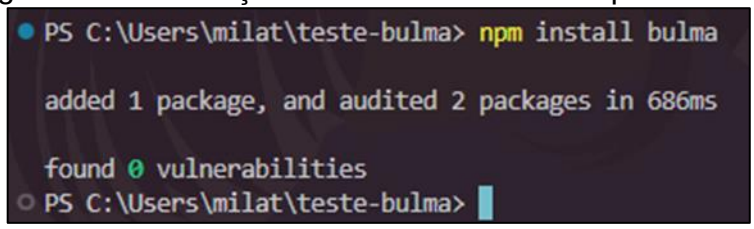
Figura 10 – Importação do Bulma feita em um arquivo HTML

Fonte: elaborado pela autora

b) baixar o arquivo compactado disponível na página do GitHub do Bulma, que contém todos os arquivos CSS e Sass necessário;

c) instalar o pacote NPM, conforme a Figura 11:

Figura 11 – Instalação do Bulma utilizando o pacote NPM



Fonte: elaborado pela autora

d) clonar o repositório do Bulma do GitHub.

Somente é possível personalizar o Bulma através da manipulação do Sass com as duas últimas formas de instalação apresentadas, pois estas possuem não somente o código CSS, como o de pré-processamento.

4.8.2 Personalizar com o pacote node-sass

Para personalizar o Bulma, é preciso seguir seis passos, que vão desde a escolha do diretório, download dos pacotes Node.js necessários à atribuição de comandos para compilar o código Sass em CSS, que estão documentados na própria página do Bulma, e que serão detalhados a seguir:

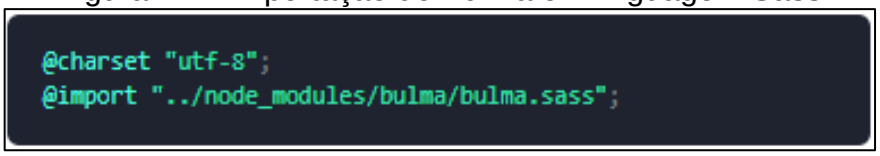
a) rodar no terminal de comando a instrução “npm init” no diretório desejado, para criar o arquivo “package.json”. Em determinado momento o terminal irá solicitar por um ponto de entrada (*entry point*), que deverá ser “sass/mystyles.scss”;

b) instalar dois pacotes do Node.js, com os comandos “npm install node-sass --save-dev” e “npm install bulma --save-dev”, para instalar, respectivamente, o pacote node-sass e o pacote do próprio Bulma;

c) criar uma pasta chamada “sass” e adicionar um arquivo chamado “mystyles.scss”, contendo a importação do Bulma em sua versão de

linguagem de pré-processamento (Figura 12), inserindo o diretório em que o pacote “bulma” foi baixado:

Figura 12 – Importação do Bulma em linguagem Sass

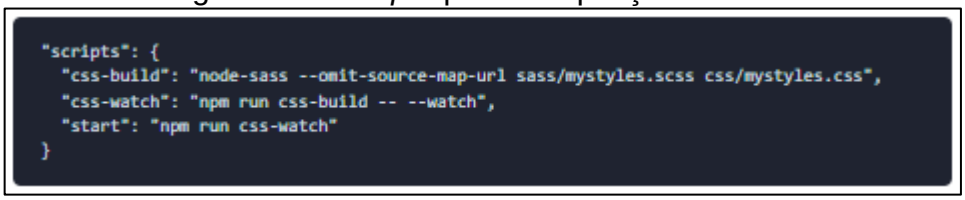


Fonte: Captura de tela realizada pela autora no site do Bulma[1]

d) criar um documento HTML que possua classes do Bulma e salvar como “mypage.html”, além de fazer a importação do arquivo CSS “mystyles.css”. Como este arquivo ainda não foi criado, pois o Sass ainda não foi compilado, a página não terá nenhum estilo até então;

e) adicionar um *script* (Figura 13) no arquivo “package.json”, para que o arquivo Sass seja compilado. Para realizar a compilação, em seguida é preciso executar o comando “npm run css-build”, presente no *script*:

Figura 13 – *Scripts* para compilação do Sass



Fonte: Captura de tela realizada pela autora no site do Bulma[2]

f) adicionar as variáveis do Bulma e seus respectivos valores antes de importar o Bulma, no arquivo “mystyles.scss”, salvar o arquivo e executar o “mypage.html” para ver a página com um estilo diferente do estilo padrão do Bulma.

4.8.3 WhatPulse

O WhatPulse é um programa gratuito (pelo menos em sua versão de teste) desenvolvido por Martijn Smit e utilizado para obter estatísticas do uso de um computador, tais como quantidade de vezes que determinado software utilizou de banda de internet, quantos cliques foram realizados ou quantas vezes o computador foi reiniciado, por exemplo (WhatPulse, 2023).

---

[1] Disponível em: <https://bulma.io/documentation/customize/with-node-sass/>. Acesso em: 23 out. 2023.
[2] *Ibid.*

A escolha do WhatPulse para adquirir os dados para os experimentos deste trabalho se deu por conta da gratuidade e facilidade de uso na aplicação, e principalmente porque o programa disponibiliza o *heatmap* (mapa de calor) do mouse, além de dados no formato “csv”, que podem ser usados em exportação para planilhas e consequentemente gerar gráficos.

Figura 14 – Tela inicial do WhatPulse

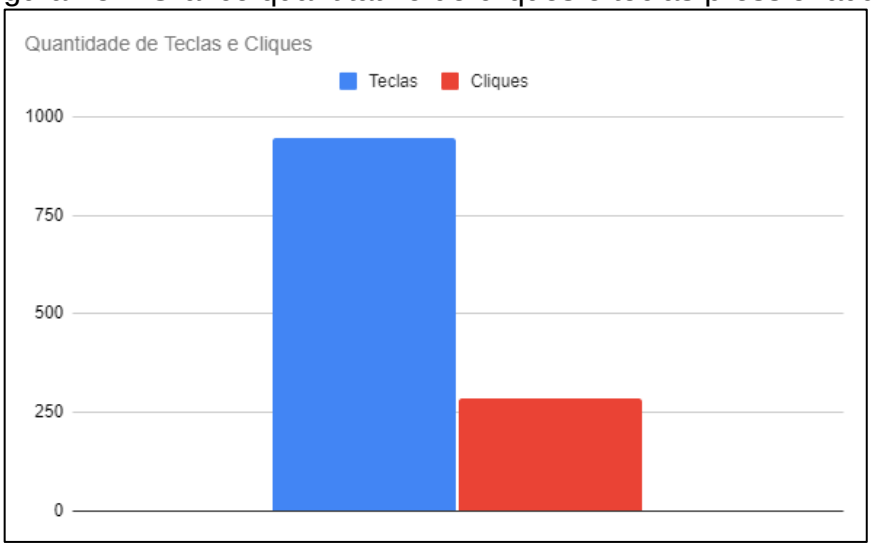
Fonte: captura de tela realizada pela autora

4.8.4 Avaliação qualitativa do modo padrão de personalização de variáveis do Bulma

A personalização do Bulma foi realizada seguindo o passo a passo explicado anteriormente na subseção 3.4.1.2. Toda a contagem de cliques e teclas se dão por todo o processo de abrir um ambiente de desenvolvimento e o navegador e acessar o site do Bulma para seguir o tutorial, para simular da forma mais fiel possível os passos que um usuário seguiria para executar o processo, que no total durou 16:40 (16 minutos e 40 segundos).

Na Figura 15, observa-se a quantidade de cliques e teclas pressionadas durante a execução das etapas. De acordo com a aplicação WhatPulse, foram 944 teclas pressionadas e 286 cliques do mouse.

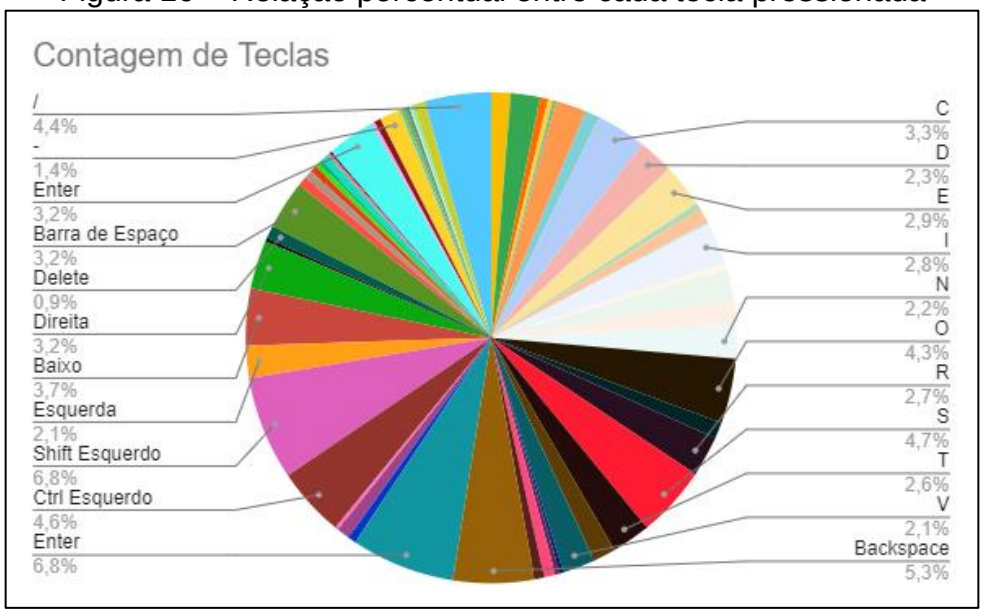
Figura 15 – Gráfico quantitativo de cliques e teclas pressionadas



Fonte: elaborado pela autora

Além disso, o subsequente gráfico (Figura 16) exibe a porcentagem de vezes que cada tecla foi pressionada; percebe-se que boa parte das teclas pressionadas, as que mais se destacam são os botões Ctrl Esquerdo (com 4,6%), Shift Esquerdo e Enter (com 6,8% cada). Estes resultados da tecla Ctrl são explicados uma vez que ocorre com frequência os atalhos de Copiar (Ctrl+C) e Colar (Ctrl+V); quanto ao Shift Esquerdo, ocorreu por conta do uso de caracteres especiais para digitar uma variável do Bulma e seu respectivo resultado, pois respectivamente, utilizam o símbolo “$” e “#” (nos códigos de cores hexadecimais); e no caso do Enter, é bastante utilizado para fazer a quebra de linha no arquivo SCSS e para confirmar instruções na linha de comando acesso em sites no navegador.

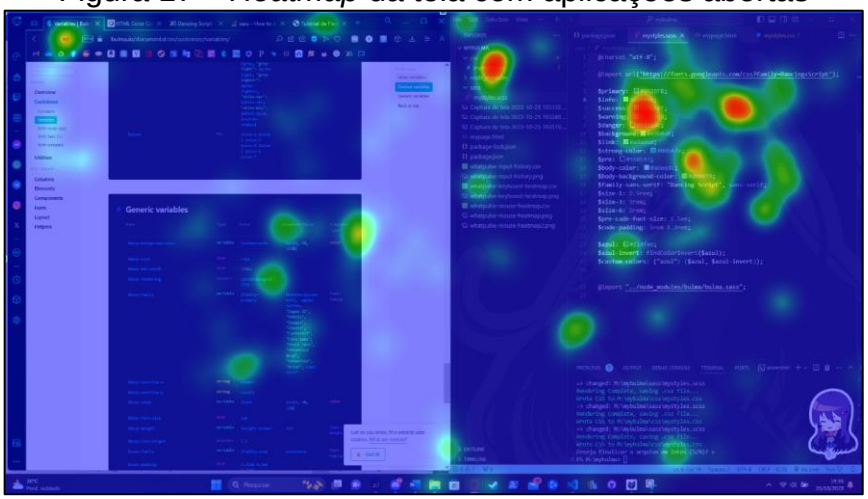
Figura 16 – Relação percentual entre cada tecla pressionada



Fonte: elaborado pela autora

O WhatPulse fornece, inclusive, *heatmap* (mapa de calor) das regiões da tela em que o mouse interagiu com cliques. A Figura 17 mostra o quanto houve interações do mouse com a IDE (lado direito), por conta da criação de arquivos, desenvolvimento do arquivo SCSS com as variáveis, execução de comandos no terminal para baixar e instalar o Bulma, enquanto do lado esquerdo, há o navegador aberto com a página do Bulma, utilizada para consultar o nome das variáveis e o respectivo tipo e unidade de cada uma e diversas interações em várias áreas da página, bem como na barra de endereços para pesquisar sobre cores hexadecimais, como utilizar a variável “$custom-colors” e acessar o próprio site do Bulma.

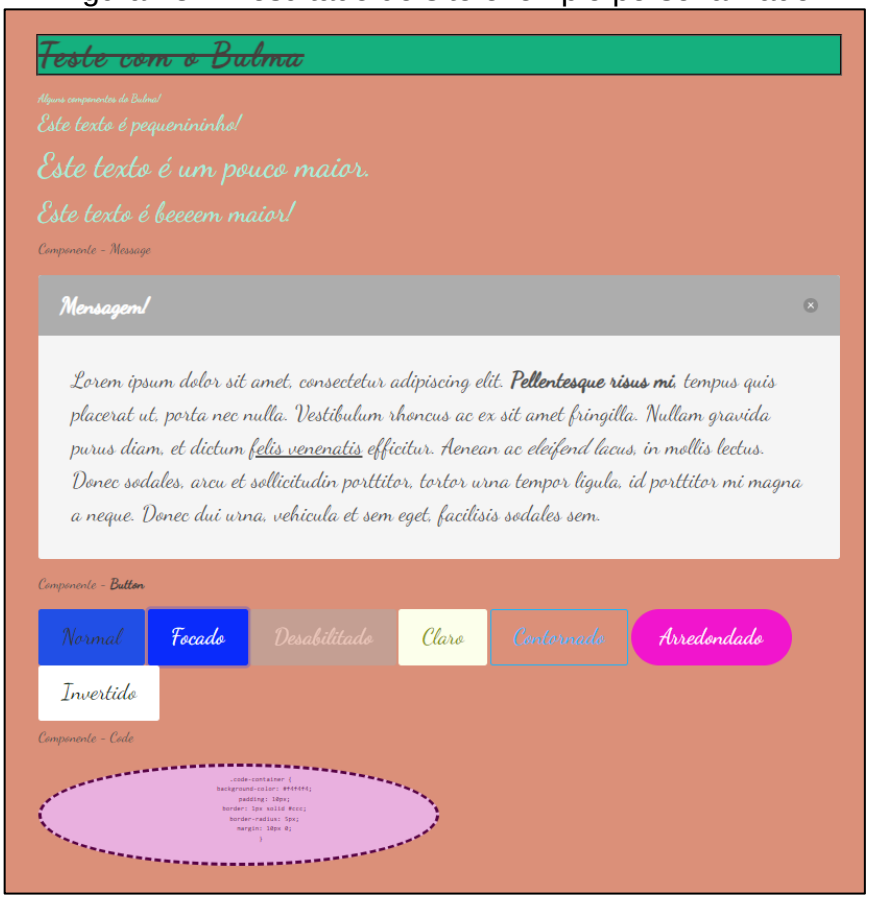
Figura 17 – *Heatmap* da tela com aplicações abertas



Fonte: elaborado pela autora

A Figura 18 mostra o resultado da personalização manual das variáveis do Bulma ao utilizar o Sass, de acordo com as especificações dos Quadros 3 e 4.

Figura 18 – Resultado do site exemplo personalizado



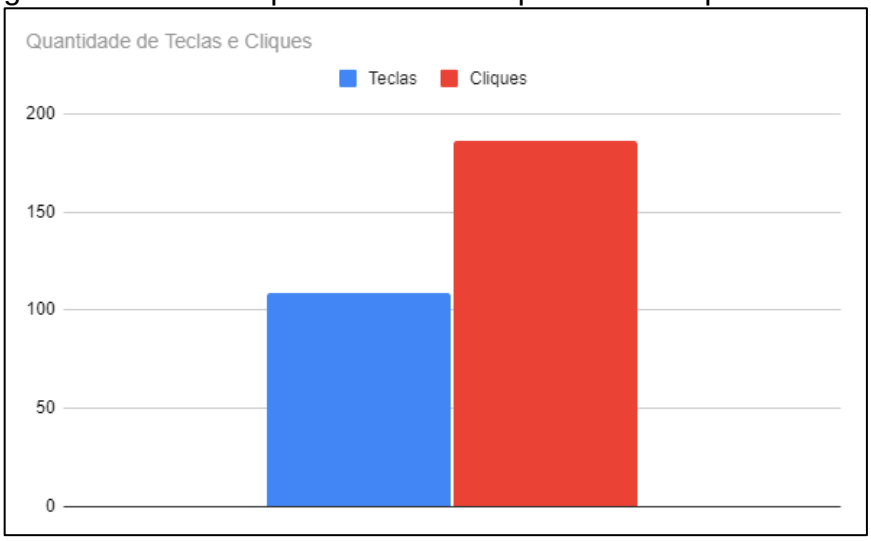
Fonte: elaborado pela autora

4.8.5 Avaliação qualitativa com a solução proposta de personalização de variáveis do Bulma

A personalização do Bulma, nesta fase do experimento, foi feita com a aplicação proposta neste trabalho. A contagem de cliques e teclas pressionadas foram realizadas da mesma forma que na personalização manual das variáveis do Bulma citada na subseção anterior. Com a solução proposta, todo o processo levou 04:50 (4 minutos e 50 segundos).

Na Figura 19, o gráfico exibe a quantidade de cliques feitos com o mouse e quantidade de teclas pressionadas no total. O WhatPulse informa que são 186 cliques e 109 teclas pressionadas.

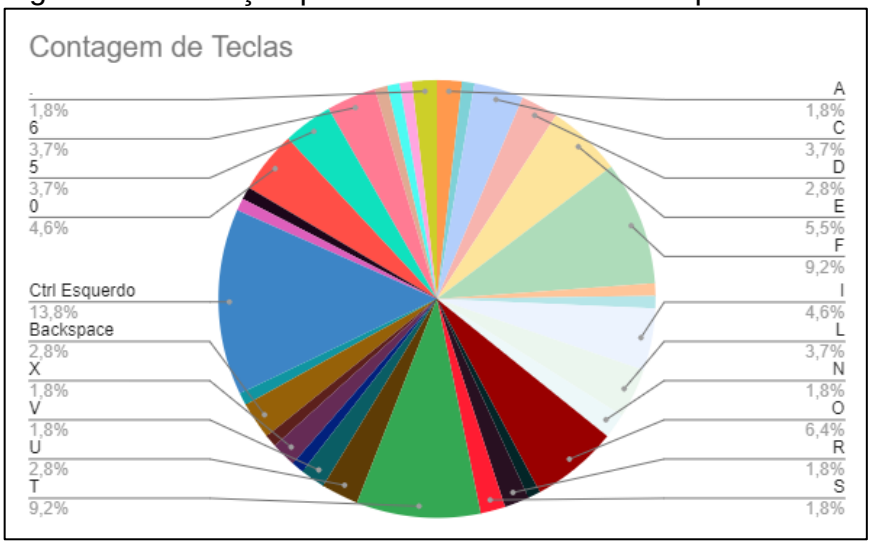
Figura 19 – Gráfico quantitativo de cliques e teclas pressionadas



Fonte: elaborado pela autora

O gráfico da Figura 20 mostra a porcentagem de vezes que cada tecla foi pressionada. As teclas que mais se destacaram foram o Ctrl Esquerdo, que se repetiu 13,8% das vezes e F, com 9,2% das vezes. A relação entre as duas se dá principalmente pelo atalho Ctrl+F, para, nesse caso, pesquisar por um atributo na tela de classes personalizáveis. As outras teclas são letras e números, digitadas não só para pesquisar por alguma palavra com o atalho Ctrl+F, quanto para nomear classes e preencher valores numéricos das variáveis.

Figura 20 – Relação percentual entre cada tecla pressionada



Fonte: elaborado pela autora

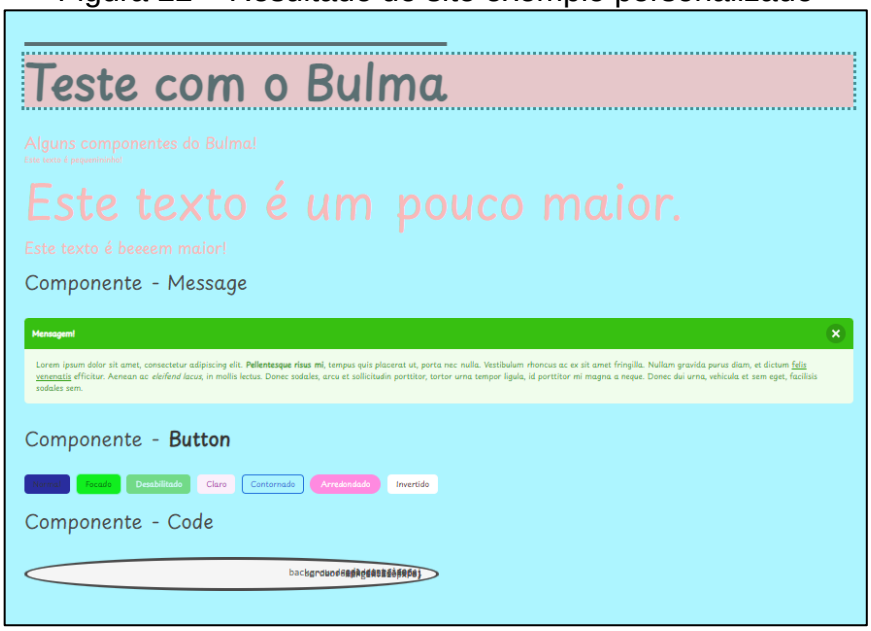
O mapa de calor da Figura 21 mostra algumas interações do lado direito da tela, na IDE, em boa parte para criar e salvar novos arquivos, além de algumas interações no código e no terminal da IDE, que fica no canto inferior direito, utilizado para iniciar o servidor. A maior parte das interações está do lado esquerdo da tela e forma bem espalhada, por conta do uso da aplicação para customizar o Bulma, tanto para selecionar o elemento a ser personalizado quanto para escolher a cor disponibilizada no seletor de cor.

Figura 21 – *Heatmap* da tela com aplicações abertas

Fonte: elaborado pela autora

A Figura 22 mostra o resultado da personalização das variáveis do Bulma, de acordo com as especificações dos Quadros 3 e 4.

Figura 22 – Resultado do site exemplo personalizado



Fonte: elaborado pela autora

4.8.6 Critérios de avaliação entre os dois modos de personalização do Bulma

Quadro 5 – Critérios de avaliação na personalização do Bulma

Critérios de avaliação	
Formato de entrada	Como é inserida ou criada a variável do Bulma para personalização.
Facilidade de uso	**Fácil**, onde não é necessário um extenso conhecimento prévio de linguagem de pré-processamento ou CSS; **médio**, no qual é preciso ter uma base de linguagem de pré-processamento ou CSS; **difícil**, quando pré-processamento ou CSS precisa ser dominado pelo usuário.
Formato de instalação/execução	Quais são os comandos necessários para executar a aplicação e compilação do código Sass.
Facilidade na instalação/execução	**Baixa**, quando o usuário não precisa ter um conhecimento de Node.js; **média**, quando o usuário entende parcialmente o que cada execução faz; **alta**, quando o usuário compreende possíveis erros de execução e consegue interpretá-los e no mínimo pesquisar como resolvê-los.
Classes personalizáveis	De que forma as classes geradas pelo usuário são criadas.
Possíveis erros	Quais erros que podem acontecer durante a compilação do código Sass.

Fonte: elaborado pela autora

4.8.7 Análise entre os experimentos

4.8.7.1 Avaliação do modo padrão de personalização do Bulma

a) Formato de entrada: totalmente manual; é necessário digitar todo o código SCSS com as variáveis do Bulma, além de desenvolver as classes personalizadas após a compilação do SCSS;
b) Facilidade de uso: médio a difícil, pois o usuário só conseguiria fazer uma classe do CSS após entender alguns conceitos de folha de estilo, bem como entender qual propriedade será modificava com a variável;
c) Formato de instalação/execução: são necessários vários comandos de execução para instalação de pacotes e compilação de códigos com o Node.js, como também de edição de arquivo CSS e do tipo JSON;
d) Facilidade na instalação/execução: difícil, uma vez que para um usuário leigo, é possível que o mesmo se depare com erros de versionamento durante a instalação de pacotes, ou não saiba onde preencher as classes CSS após a compilação do código;
e) Classes personalizáveis: geradas manualmente e somente após a compilação do SCSS;
f) Possíveis erros: erros de versionamento de pacote Node.js, ordem incorreta de elementos do arquivo SCSS ou CSS.

4.8.7.2 Avaliação da personalização do Bulma utilizando a solução proposta

a) Formato de entrada: por cliques no mouse ou entradas no teclado, ao escolher as cores pelos seletores ou ao interagir com os formulários de entradas numéricas;
b) Facilidade de uso: fácil, pois a aplicação fornece os tipos de cada variável ou atributo do CSS, bem como os possíveis elementos aceitos por cada atributo através de um menu expansivo;
c) Formato de instalação/execução: o único comando necessário é o de iniciar o servidor;

d) Facilidade na instalação/execução: fácil, já que uma vez que o único comando que o usuário precisa realizar é o de iniciar o servidor, as outras etapas são apenas de interação com formulários na página da aplicação, e o usuário pode ficar sem se preocupar em como a compilação será realizada;
e) Classes personalizáveis: são geradas ao interagir com a aplicação e estas classes são adicionadas automaticamente após a compilação do código CSS;
f) Possíveis erros: toda a aplicação filtra e manipula os dados de modo em que o usuário só possa inserir dados válidos para os respectivos atributos e nomes de classes do CSS e das variáveis do Bulma, portanto o único erro possível seria a compilação do Sass com o servidor desligado.

4.8.7.3 Tabela comparativa entre os experimentos realizados

Tabela 2 – Comparação entre os experimentos

	Sem a solução proposta	**Com a solução proposta**
Formato de entrada	Manual	Interação com o formulário
Facilidade de uso	Médio a difícil	Fácil
Formato de instalação/execução	Comandos de instalação e compilação	Iniciar o servidor
Facilidade na instalação/execução	Difícil	Fácil
Classes personalizáveis	Manual	Interação com o formulário
Possíveis erros	Versionamento de pacote Node.js, formatação incorreta do SCSS ou CSS	Não funciona com o servidor desligado

Fonte: elaborado pela autora

5 CONCLUSÃO

Este trabalho possibilitou o estudo do estado da arte em padrões de projeto para o desenvolvimento web para criar uma aplicação que construísse de forma automatizada um arquivo CSS através da compilação de um arquivo SCSS, e, por fim, a análise dos resultados obtidos através desse aplicativo desenvolvido. A principal motivação da aplicação desenvolvida neste projeto seria para o uso de alunos da disciplina de Programação para Web, ministrada pelo professor Leandro José Komosinski, que sempre utilizava o Bulma em suas aulas.

A aplicação proposta neste trabalho visou facilitar a customização das variáveis do *framework* Bulma e pode ser utilizada em qualquer página web que utilize o Bulma, uma vez que o CSS gerado é compilado utilizando a biblioteca “node-sass” e “bulma” do Node.js, ou seja, utilizando as mesmas bibliotecas caso a customização das variáveis ocorresse de forma manual, seguindo o tutorial da documentação oficial disponível na página do Bulma.

De acordo com o objetivo geral, a solução desenvolvida mostrou-se mais eficaz que o modo padrão de personalização do Bulma: no experimento realizado neste trabalho, com o WhatPulse, apontou que a customização das variáveis mais a criação de classes de CSS foram 71% mais rápidas utilizando a aplicação, além de ter cerca de 35% menos cliques e 88% menos teclas digitadas que o modo convencional.

Desenvolvedores que estão aprendendo sobre *front-end* se beneficiam do uso desta aplicação não só pela facilidade do uso, mas principalmente pela instalação e execução dos pacotes e comandos Node.js. Porém, isso não impede que os desenvolvedores mais experientes aproveitem desses benefícios da mesma forma. Qualquer aplicação web que utilize folhas de estilo pode se usufruir do arquivo CSS gerado, independentemente do tamanho ou dos *frameworks* utilizados.

Para deixar a aplicação ainda mais completa e funcional, principalmente para um uso coletivo em sala de aula, como era a motivação inicial deste projeto, futuramente poderia ser implementada a funcionalidade de persistir em banco de dados cada customização realizada, de forma que a mesma pudesse ser acessada por outros usuários, bem como fazer o processo de engenharia reversa e fazer upload de um arquivo CSS baixado para ser editado novamente, além de fazer a aplicação

funcionar em um servidor *online*, para que não seja de uso apenas local. Além disso, a solução proposta se beneficiaria com uma otimização do código para que a ordenação dos formulários de atributos do CSS e variáveis do Bulma sejam feitos de forma dinâmica e automatizada, tornando assim a página mais rápida, bem como adicionar todas as variáveis do Bulma e todos os elementos CSS na aplicação.

REFERÊNCIAS

ABOUT US | WhatPulse. Disponível em: https://whatpulse.org/about/. Acesso em: 25 out. 2023.

ANTUNES, Hiuram; FONSECA, Inacio de Sousa Adelino da. Advanced web methodology for flexible web development. **2021 16Th Iberian Conference On Information Systems And Technologies (Cisti)**, [S.L.], v. 3, p. 1-4, 23 jun. 2021. IEEE. http://dx.doi.org/10.23919/cisti52073.2021.9476295.

BŁASZYŃSK, Łukasz. **Front-end history, 2018 trends and Espeo choices**. 2018. Disponível em: https://espeo.eu/blog/front-end-history-2018-trends-and-espeo-choices/. Acesso em: 09 jul. 2021.

BERKOVSKYY, Andrii; VOSKOBOINIK, Kostiantyn; BADUROWICZ, Marcin. Comparison of the compilation speed of the SCSS and LESS preprocessors. **Journal Of Computer Sciences Institute**, [S.L.], v. 20, p. 225-229, 30 set. 2021. Politechnika Lubelska. http://dx.doi.org/10.35784/jcsi.2692.

BROOKS, David R.. **An Introduction to HTML and JavaScript**: for scientists and engineers. [S. L.]: Springer, 2007.

BULMA: Free, open source, and modern CSS framework based on Flexbox. 2021. Disponível em: https://bulma.io. Acesso em: 11 set. 2021.

DELCEV, Sanja; DRASKOVIC, Drazen. Modern JavaScript frameworks: a survey study. **2018 Zooming Innovation In Consumer Technologies Conference (Zinc)**, [S.L.], v. 1, p. 106-109, maio 2018. IEEE. http://dx.doi.org/10.1109/zinc.2018.8448444.

FENDER, Joe; YOUNG, Carwin. **Front-End Fundamentals**: a practical guide to front-end web development.. [S. L.]: Leanpub, 2015.

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. **Design Patterns**: elements of reusable object-oriented software. Indianapolis, In: Pearson, 1994.

GARCIA-LUNA-ACEVES, J.J.; ALBALAWI, Abdulazaz Ali. Connection-Free Reliable and Efficient Transport Services in the IP Internet. **2020 16Th International Conference On Network And Service Management (Cnsm)**, [S.L.], v. 1, n. 1, p. 1-1, 2 nov. 2020. IEEE. http://dx.doi.org/10.23919/cnsm50824.2020.9269115.

GOSWAMI, Pronnoy; GUPTA, Saksham; LI, Zhiyuan; MENG, Na; YAO, Daphne. Investigating The Reproducibility of NPM Packages. **2020 Ieee International Conference On Software Maintenance And Evolution (Icsme)**, [S.L.], v. 36, p. 677-681, set. 2020. IEEE. http://dx.doi.org/10.1109/icsme46990.2020.00071.

KAUSHAL, Utkarsh; SINGH, Gurinder; PARASHAR, Tarun. Responsive Webpage Using HTML CSS. **2022 International Conference On Cyber Resilience (Iccr)**, [S.L.], v. 1, p. 1-4, 6 out. 2022.
IEEE. http://dx.doi.org/10.1109/iccr56254.2022.9995922.

LAUKKANEN, Toni Harald Antero. **CSS preprocessor – Sass eller Less**. 2017. 37 f. TCC (Graduação) - Curso de Information And Media Technology, Arcada University Of Applied Sciences, Helsinque, 2017.

MAZINANIAN, Davood; TSANTALIS, Nikolaos. An Empirical Study on the Use of CSS Preprocessors. **2016 Ieee 23Rd International Conference On Software Analysis, Evolution, And Reengineering (Saner)**, [S.L.], v. 2, p. 168-178, mar. 2016. IEEE. http://dx.doi.org/10.1109/saner.2016.18.

MAZINANIAN, Davood; TSANTALIS, Nikolaos. CSSDev: refactoring duplication in cascading style sheets. **2017 Ieee/Acm 39Th International Conference On Software Engineering Companion (Icse-C)**, [S.L.], v. 1, p. 63-66, maio 2017. IEEE. http://dx.doi.org/10.1109/icse-c.2017.7.

MESBAH, Ali; MIRSHOKRAIE, Shabnam. Automated analysis of CSS rules to support style maintenance. **2012 34Th International Conference On Software Engineering (Icse)**, Zurique, v. 1, n. 1, p. 408-418, jun. 2012. IEEE. http://dx.doi.org/10.1109/icse.2012.6227174.

MONTEIRO, Rui Tiago Bugalho. **Arquitetura Orientada a Componentes para uma Web Responsiva**. 2015. 76 f. Dissertação (Mestrado) - Curso de Engenharia Informática e Computação, Faculdade de Engenharia da Universidade do Porto, Porto, Portugal, 2015.

PFAFFENBERGER, Brian; SCHAFER, Steven M.; WHITE, Charles; KAROW, Bill. **HTML, XHTML, and CSS Bible**. 3. ed. Indianapolis, In: Wiley, 2004.

PRABHU, Anirudh. **Beginning CSS Preprocessors**: with sass, compass, and less. [S. L.]: Apress, 2015.

QUEIRÓS, Ricardo. A Survey on CSS Preprocessors. **Schloss Dagstuhl - Leibniz-Zentrum Fuer Informatik Gmbh, Wadern/Saarbruecken, Germany**, [S. L.], v. 56, p. 1-12, 04 out. 2017. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik GmbH, Wadern/Saarbruecken, Germany. http://dx.doi.org/10.4230/OASICS.SLATE.2017.8.

SASS: Documentation. 2013. Disponível em: https://sass-lang.com/documentation. Acesso em: 10 set. 2021.

WANG, P.P.; WEI, Z.Q.; LI, Z.. Front-end Website Performance Optimization Based on Web Storage in Html5. **Software Engineering And Information Technology**, Qingdao, p. 36-40, 17 dez. 2015. WORLD SCIENTIFIC. http://dx.doi.org/10.1142/9789814740104_0008.

WITH node-sass | Bulma: Free, open source, and modern CSS framework based on Flexbox. Disponível em: https://bulma.io/documentation/customize/with-node-sass/. Acesso em: 23 out. 2023.

WITTERN, Erik; SUTER, Philippe; RAJAGOPALAN, Shriram. A look at the dynamics of the JavaScript package ecosystem. **Proceedings Of The 13Th International Conference On Mining Software Repositories**, [S.L.], v. 13, p. 351-361, 14 maio 2016. ACM. http://dx.doi.org/10.1145/2901739.2901743.

APÊNDICE A – Repositório da aplicação

A aplicação desenvolvida neste trabalho pode ser encontrada no seguinte repositório do GitHub: https://github.com/MilaKings/BulmaCustomizer

APÊNDICE B – Artigo do TCC

Ambiente de desenvolvimento integrado para personalização automatizada do framework Bulma mediante variáveis do Sass

Camila dos Reis[1]

[1]Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
Caixa Postal 476 – 88.010-970 – Florianópolis – SC – Brazil

milareis@grad.ufsc.br

Abstract. *Developers started to use preprocessors, filling the logic functions missing on CSS. The present academic work aims to develop a model to automate the CSS file creation through preprocessing. To generate this file, the user access an specific web page that works like an integrated development environment and can customize Bulma framework, with no need of previous knowledge in preprocessing.. At the end of this paper, a comparison is made between the customization of Bulma with and with and without the application developed, which proved to be much faster and simpler to use.*

Resumo. *Para suprir a carência de operações lógicas do CSS, os desenvolvedores passaram a adotar o uso de pré-processadores. O presente trabalho tem como objetivo propor um modelo para automatizar a criação de um arquivo CSS através da adoção de pré-processamento, com uma página web que funciona como um ambiente de desenvolvimento integrado de forma que seja possível personalizar o framework Bulma, sem a necessidade de ter conhecimentos prévios sobre pré-processamento. Por fim, é realizada uma comparação entre a personalização do Bulma com e sem a aplicação desenvolvida, e a aplicação deste TCC se mostrou muito mais rápida e simples de utilizar.*

1. Introdução

Graças à ascensão de tecnologias web e equipamentos computacionais, foi possível criar páginas web mais complexas e com mais funcionalidades. Com isso, tornou-se necessário criar a classificação back-end, para desenvolvimento da parte do servidor, e front-end, para o lado do cliente. É na programação do front-end que se desenvolve como a página web se comporta para que o usuário possa interagir [Fender; Young, 2015].

Este artigo propõe facilitar o emprego de pré-processadores (mais especificamente o Sass, uma linguagem de pré-processamento, que concede o uso de operadores numéricos, lógicos, de igualdade, funções, entre outros, e ao ser compilada, se torna um código em CSS [Sass: Syntactically Awesome Style Sheets, 2013]), mediante a automatização de um arquivo CSS (especificamente do framework Bulma, um framework voltado para auxiliar na produção de uma página web, com componentes responsivos de CSS [Bulma, 2021]).

Essa personalização do Sass em uma página web, que atuará como um ambiente de desenvolvimento integrado, segue o padrão de projeto MVC (Model-View-Controller), um tipo de padrão de projeto dividido em três classes: Model (Modelo), que trata dos dados da aplicação, como questões de armazenamento e criação de dados; View (Visão), na qual representa a interface do usuário e Controller (Controlador), que funciona como uma ponte entre Model e View, pois é a parte responsável por obter os dados recebidos pela Visão e

repassá-los para o Modelo, e vice-versa [Gamma; Helm; Johnson; Vlissides, 1994]. Além disso, neste ambiente de desenvolvimento será 18 possível aplicar o CSS gerado na própria página, para prova de conceito, de forma que seja possível visualizar o resultado, realizar o download do CSS e criar classes de CSS personalizadas, além das já existentes no Bulma.

2. Objetivos

Propor e analisar um novo modelo para automatizar a criação de um arquivo CSS mediante adoção de pré-processamento, além de analisar o estado da arte em padrões de projeto para desenvolvimento web e abordagens específicas para front-end e utilizando o resultado da análise realizada, desenvolver um modelo para a automatização de CSS baseado em pré-processamento. Por fim, analisar o modelo mediante um ambiente de desenvolvimento integrado para obter a prova de conceito.

3. Prototipação

Durante a fase de prototipação da aplicação, foi desenvolvida a tela inicial, contendo duas abas: "Variáveis", onde é possível modificar as variáveis do Bulma e "Criar Classes", para gerar classes personalizadas. Na primeira aba, como pode-se observar na Figura 1, possui 4 das 419 variáveis do Bulma.

Seguindo a ordem da esquerda para a direita, o usuário poderá escolher uma fonte do Google Fonts e colar a URL no campo "@import url", para realizar a importação da fonte e incorporá-la ao arquivo CSS compilado. O formato da URL seria algo semelhante a "https://fonts.googleapis.com/css?family=Handlee", onde "Handlee" seria uma das fontes disponibilizadas no Google Fonts.

Os próximos três elementos são botões seletores de cor, estilizados com CSS para obterem esta aparência circular. Na Figura 2, a cor da variável "$primary" está sendo modificada. Quando o botão "Compilar Sass" é clicado, respectivamente "Baixar CSS" e "Testar CSS" passam a ser habilitados, permitindo então o download do arquivo CSS e também um teste deste arquivo recém compilado. Como o projeto também utiliza o Bulma para estilização, testar este CSS gerado irá alterar os elementos "$primary", "$link", "$body-background-color" e a fonte, como é possível verificar na Figura 3.

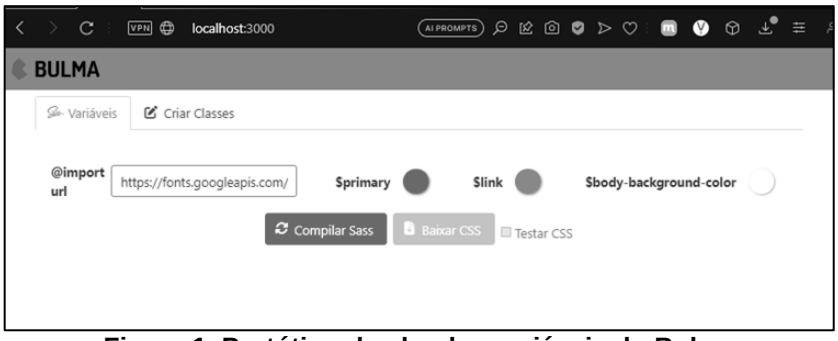


Figura 1. Protótipo da aba das variáveis do Bulma

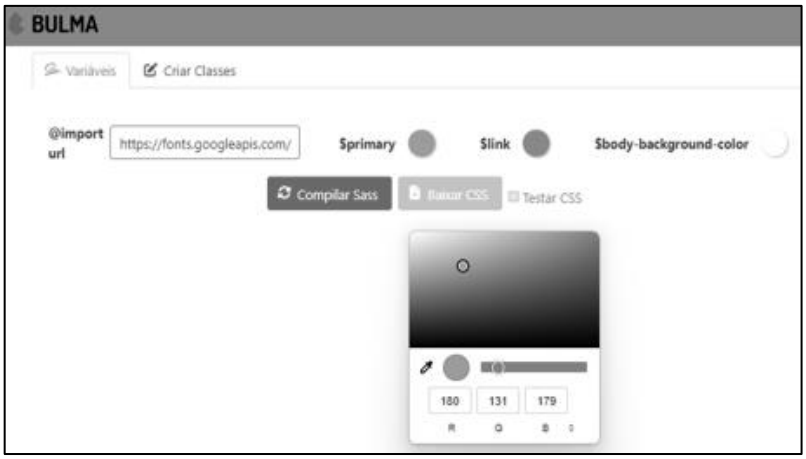


Figura 2. Seletor de cores da variável $primary

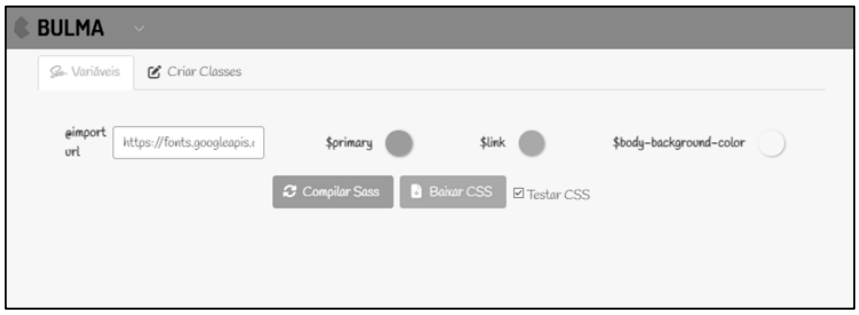


Figura 3. Personalização do Bulma em tempo real

4. Experimentos

Esta seção realiza dois experimentos para verificar as diferenças tanto do modo padrão de personalização do Bulma quanto com a solução proposta por este trabalho. Para verificar esta diferença, é comparado o passo a passo da instalação e execução da personalização das variáveis, com e sem a aplicação desenvolvida, com métricas de tempo de duração, quantidade de cliques de caracteres digitados para personalização do framework Bulma em um mesmo site e com as mesmas variáveis e mapa de calor da tela. Todo este processo é feito graças à aplicação WhatPulse [WhatPulse, 2023], que permite monitorar cliques e teclas de um computador, além de gerar tabelas com dados. Além disso, também é analisado o uso da aplicação e sem para criar novas classes para o CSS.

5. Avaliação do modo usual de personalização

A personalização do Bulma foi realizada seguindo o passo a passo exemplificado no website do Bulma. Toda a contagem de cliques e teclas se dão por todo o processo de abrir um ambiente de desenvolvimento e o navegador e acessar o site do Bulma para seguir o tutorial, para simular da forma mais fiel possível os passos que um usuário seguiria para executar o processo, que no total durou 16:40 (16 minutos e 40 segundos). A quantidade de cliques e teclas pressionadas durante a execução das etapas. De acordo com a aplicação WhatPulse, foram 944 teclas pressionadas e 286 cliques do mouse.

6. Avaliação com a aplicação proposta para a personalização

A personalização do Bulma, nesta fase do experimento, foi feita com a aplicação proposta neste trabalho. A contagem de cliques e teclas pressionadas foram realizadas da mesma forma que na personalização manual das variáveis do Bulma citada na seção anterior. Com a solução proposta, todo o processo levou 04:50 (4 minutos e 50 segundos). O WhatPulse informa que

são 186 cliques e 109 teclas pressionadas.

7. Conclusão

Este trabalho possibilitou o estudo do estado da arte em padrões de projeto para o desenvolvimento web para criar uma aplicação que construísse de forma automatizada um arquivo CSS através da compilação de um arquivo SCSS, e, por fim, a análise dos resultados obtidos através desse aplicativo desenvolvido.

A aplicação proposta neste trabalho visou facilitar a customização das variáveis do framework Bulma e pode ser utilizada em qualquer página web que utilize o Bulma, uma vez que o CSS gerado é compilado utilizando a biblioteca “node-sass” e “bulma” do Node.js, ou seja, utilizando as mesmas bibliotecas caso a customização das variáveis ocorresse de forma manual, seguindo o tutorial da documentação oficial disponível na página do Bulma. De acordo com o objetivo geral, a solução desenvolvida mostrou-se mais eficaz que o modo padrão de personalização do Bulma: no experimento realizado neste trabalho, com o WhatPulse, apontou que a customização das variáveis mais a criação de classes de CSS foram 71% mais rápidas utilizando a aplicação, além de ter cerca de 35% menos cliques e 88% menos teclas digitadas que o modo convencional.

8. Referências

ABOUT US | WhatPulse. Disponível em: https://whatpulse.org/about/. Acesso em: 25 out. 2023.

BULMA: Free, open source, and modern CSS framework based on Flexbox. 2021. Disponível em: https://bulma.io. Acesso em: 11 set. 2021.

FENDER, Joe; YOUNG, Carwin. Front-End Fundamentals: a practical guide to front-end web development.. [S. L.]: Leanpub, 2015.

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. Design Patterns: elements of reusable object-oriented software. Indianapolis, In: Pearson, 1994.

SASS: Documentation. 2013. Disponível em: https://sass-lang.com/documentation. Acesso em: 10 set. 2021.