UNIVERSIDADE FEDERAL DO RIO GRANDE - FURG
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E
DESENVOLVIMENTO DE SISTEMAS

GUILHERME CORRÊA MORGADO

SASS4J: UM COMPONENTE SASS PARA A PLATAFORMA JAVA EE

Trabalho de Conclusão de Graduação

Prof. Esp. Márcio Josué Ramos Torres
Orientador

Rio Grande, Julho de 2014

FOLHA DE APROVAÇÃO

Monografia sob o título "SASS4J: UM COMPONENTE SASS PARA A PLATAFORMA JAVA EE" defendida por GUILHERME CORRÊA MORGADO e aprovada em 10 de Julho de 2014, em Rio Grande, estado do Rio Grande do Sul, pela banca examinadora constituída pelos professores:

_____

Prof. Esp. Márcio Josué Ramos Torres
Orientador

_____

Prof. Ms. Cibele da Rosa Christ Sinoti
IFRS - *Campus* Rio Grande

_____

Prof. Dr. Tiago Lopes Telecken
IFRS - *Campus* Rio Grande

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

ANTLR	ANother Tool for Language
API	Application Programming Interface
ASF	Apache Software Foundation
CSS	Cascading Style Sheets
DSL	Domain-specific Language
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
jRuby	The Ruby Programming Language on the JVM
Java EE	Java Enterprise Edition
JSP	JavaServer Pages
JVM	Java Virtual Machine
MIME	Multipurpose Internet Mail Extensions
Sass	Syntactically Awesome Stylesheets
WWW	World Wide Web
W3C	World Wide Web Consorcium

LISTA DE FIGURAS

RESUMO

O componente sass4j foi desenvolvido com a intuição de ajudar o desenvolvimento *front-end*, especificamente utilizando Sass em sua sintaxe Scss, em uma plataforma de desenvolvimento conceituada e que não oferecia suporte a esta linguagem. Além de possibilitar um método de usar Sass com Java EE de uma maneira simples, também foi levado em consideração a contribuição a todos que possam precisar deste mesmo componente, visto que Sass é um dos maiores preprocessadores de CSS existentes no mercado. Mediante o uso de jRuby para conectar o compilador já existente do Sass com a plataforma Java EE, o tratamento dos arquivos a serem processados foi realizado inteiramente na aplicação Java, resultando em uma aplicação com capacidade de integração em qualquer projeto que utilize Java EE.

Palavras-chave: Sass, Java EE, jRuby.

SASS4J: A Component Sass for Java EE Platform

ABSTRACT

The sass4j component was developed with the intuition to help front-end development, specifically using Sass in SCSS syntax, a platform for development and conceptualized that did not support this language. Besides enabling a method to use Sass with Java EE in a simple way, was also taken into account the contribution to all who may need this same component, since Sass is one of the largest CSS preprocessors on the market. By using jRuby to connect the existing Sass compiler with the Java EE platform, the processing of files to be processed was conducted entirely in the Java application, resulting in an application with the ability to integrate into any project that uses Java EE.

Keywords: Sass, Java EE, jRuby.

1 INTRODUÇÃO

Quando o desenvolvedor está escrevendo um código *front-end* para um novo projeto, há a repetição de algumas ações básicas desse cenário, como a criação de novas classes de estilos e reutilização de números para medidas. Este tempo gasto com repetições cresce de acordo com o tamanho do projeto, ou seja, quanto maior o número de páginas a serem desenvolvidas, maior será o número de repetições a serem feitas. Com o intuito de reduzir este tempo de repetições, foram criadas ferramentas para auxiliar o desenvolvimento de código, chamadas preprocessadores de CSS (*Cascading Style Sheets*).

Conforme Libby (2013), esses preprocessadores recebem instruções para facilitar a repetição de códigos, a edição ou até o reaproveitamento de parte de um código. Sendo assim, podendo ser aproveitado um trecho quantas vezes forem necessárias, a operação de edição de um código se torna mais fácil e até o reaproveitamento de parte de um código, para ser utilizado em outro módulo, se torna uma operação menos propensa a erros.

O projeto visa ajudar o desenvolvedor que utiliza estes preprocessadores, visto que estas ferramentas não possuem saídas existentes em todas as linguagens de programações. O projeto se específica ao criar um componente para a plataforma Java EE (Java *Enterprise Edition*) de um preprocessador já escolhido: o Sass (*Syntactically Awesome Stylesheets*), efetuando o processamento do Sass nativo utilizando a biblioteca jRuby. Para com isso, colaborar com a produtividade dos desenvolvedores que utilizem a combinação desta ferramenta com a linguagem escolhida.

1.1 Objetivos

O objetivo do projeto é analisar e desenvolver um componente Java EE utilizando

jRuby para processar o arquivos escritos com a sintaxe Scss.

1.1.1 Objetivos específicos

Para atingir esse objetivo, podem se enumerar os seguintes objetivos específicos:

- Analisar e desenvolver um módulo Java EE utilizando filtros;
- Analisar e portar o módulo já existente do Sass, escrito em Ruby, para a plataforma Java EE utilizando jRuby;
- Analisar e desenvolver um mecanismo de "cacheamento" de arquivos manipulados pelo componente criado.

1.2 Motivação

A motivação para realizar este projeto partiu no momento em que observou-se o tempo que era gasto com operações simples, como edição de códigos CSS. Percebeu-se que o tempo gasto copiando e colando códigos que já estavam prontos, estava excessivamente longa. Assim, pensou-se sobre a possibilidade de haver otimização do tempo. Ao ser orientado, foi possível integrar a necessidade de um desenvolvedor *front-end*, com a lacuna que há entre este preprocessador e a plataforma de programação utilizada.

1.3 Para que serve

Um preprocessador de CSS é um conversor de texto baseado em conversões léxicas. Esta é uma forma mais produtiva de desenvolver CSS. O Sass além de fornecer uma forma mais simples e produtiva de escrever CSS, através da sua própria metalinguagem, possibilita a criação de arquivos separados, organizados, que após serem compilados, formam arquivos CSS. Além de organização e produção, Sass garante a criação de funções, uso de variáveis e expressões matemáticas. Hoje, o Sass é compilado somente via Ruby. A utilidade deste projeto é criar uma versão do compilador para aplicações escritas em Java, adaptando a aplicação Sass já existente, escrita em Ruby, para que aplicações Java EE possam utilizar esse preprocessador e haja um ganho em tempo de produção.

1.4 Público alvo

O público alvo é toda a comunidade desenvolvedora que utiliza o Sass como seu preprocessador de CSS e utiliza a plataforma Java EE como plataforma de desenvolvimento.

1.5 Resumo dos capítulos

O trabalho é organizado e divido em capítulos, descritos do seguinte modo:

- Capítulo 2: Trata de Web e suas bases. Este capítulo serve de suporte do trabalho que é baseado em estilização de elementos apresentados na interface Web.

- Capítulo 3: Fala sobre Preprocessadores de CSS. A finalidade da explicação dos preprocessadores é devido a utilização e análise das ferramentas existentes no mercado, e também da seleção de uma ferramenta para execução do projeto.

- Capítulo 4: Trata da Plataforma Java EE, a plataforma escolhida como ambiente de desenvolvimento do projeto. Também fala sobre os componentes da plataforma utilizados no projeto.

- Capítulo 5: Trata de jRuby como camada de conexão entre a plataforma Java EE e o compilador Sass. Também é citada a linguagem de programação Ruby, para que tenha-se compreensão de como é feito o compilador Sass.

- Capítulo 6: Trata do sass4j, o componente desenvolvido neste projeto. Dentro deste capítulo serão tratados todos os pontos de análise e execução do projeto.

- Capítulo 7: Trata da conclusão do desenvolvimento e de ideias futuras relacionadas ao projeto.

2 WEB

Neste capítulo, serão abordados os assuntos que foram utilizados durante o desenvolvimento do projeto, para melhor compreensão de todos.

Teia mundial, tradução de World Wide Web, são documentos que são manipulados e executados na Internet. Para a visualização destes documentos, são utilizados navegadores, que por si descarregam essas informações de servidores para mostrar o resultado ao usuário. A Web segue o preceito de navegação, o qual diz que o usuário tem a possibilidade de seguir um fluxo de páginas por meio de interligações entre as mesmas.

2.1 Protocolo HTTP

HTTP (*HyperText Transfer Protocol*), em português, Protocolo de Transferência de Hipertexto, é a base de comunicação de todas as informações que passam pela Web. Esse protocolo garante que todas as informações transitem pela Web, através de requisições e respostas, sem falhas ou danos causados por terceiros. (GOURLEY et al., 2002).

O protocolo é dividido em diversos métodos através de um modelo computacional cliente-servidor, ou de uma forma mais clara, requisição-resposta. Este protocolo é usado pela WWW desde 1990, e desde lá, já foram publicadas duas versões até existir a que persiste até hoje, a versão HTTP/1.1.

Em toda a internet, há milhares de tipos de arquivos, e para o protocolo HTTP não ter falhas durante seu processo de entrega, estes são marcados com um rótulo chamado MIME (*Multipurpose Internet Mail Extensions*), em português, Extensões Multifunção para Mensagens de Internet.

Para que todas as informações que circulam na Web funcionem perfeitamente, os

servidores Web adicionam a todos objetos HTTP este rótulo, de forma que quando o cliente receba o objeto enviado, possa interpretar e manusear corretamente o objeto.

2.2 HTML

HTML (*HyperText Markup Language*), em português, Linguagem de Marcação Hipertexto, é a linguagem de marcação mais utilizada para criação de paginas Web. O HTML é mundialmente conhecido por sua compatibilidade com navegadores de internet. O documento é escrito usando *tags* pré-definidas pela linguagem, e suas especificações são escritas atualmente pelo W3C (*World Wide Web Consorcium*), órgão que padroniza a Web.

HTML consiste em escrever elementos que ao se comunicarem com o navegador, criem informações baseadas no conteúdo escrito no elemento. Os elementos normalmente possuem uma *tag* de abertura e uma para o seu fechamento (a de fechamento contém uma barra extra no começo da mesma). (DUCKETT, 2011).

Para estilizar *tags* HTML, são usadas classes CSS, como mostra a Figura 2.1.

```
<section id="canais-atendimento">
    <div class="wrapper">
        <div class="col-canais first">
            <h6 title="Canais De Atendimento">Canais De Atendimento:</h6>
        </div>
        <div class="col-canais chatonline">
            <a class="ico ico-chat-online" href="javascript:void(0);" title="Chat On-line">Chat On-line</a>
        </div>
        <div class="col-canais">
            <a class="ico ico-confira-telefones" href="javascript:void(0);" title="Confira Os Telefones">Confira os Telefones</a>
        </div>
        <div class="col-canais">
            <a class="ico ico-envie-mensagem" href="javascript:void(0);" title="Envie Uma Mensagem">Envie uma Mensagem</a>
        </div>
        <div class="col-canais last">
            <a class="ico ico-unidades-venda" href="javascript:void(0);" title="Unidades De Venda">Unidades de Venda</a>
        </div>
        <br class="clr" />
    </div>
</section>
```

Figura 2.1: Exemplo de código HTML com classes CSS.

Fonte: Autoria própria.

2.3 JavaScript

JavaScript é uma linguagem de programação interpretada pelos navegadores. É usada como uma linguagem de programação de utilização tanto do lado do cliente, quanto do servidor, porém sua maior utilização é do lado do cliente, para que os *scripts* criados não necessitem ser processados pelo servidor.

JavaScript possui algumas características notáveis, como orientação a objetos (baseado em protótipos), tipagem dinâmica e funções de primeira classe. Essas características que garantem o sucesso que JavaScript possui na rotina profissional, visto que a maioria dos navegadores e aplicativos que são utilizados para o consumo de informações da Web, possuem suporte a JavaScript. (FLANAGAN; MATSUMOTO, 2011).

Culturalmente, JavaScript também é conhecido como a linguagem que define o comportamento das páginas Web, visto que há um enorme conjunto de bibliotecas especializados em facilitar operações de controle de dados informados e criação de comportamentos visuais na tela.

Na Figura 2.2, pode ser visto um pequeno trecho de código JavaScript utilizando uma das bibliotecas mais usadas, o jQuery.

```
1   $(document).ready(function(){
2
3       var nav = $('.nav-container');
4
5       $(window).scroll(function () {
6           if ($(this).scrollTop() > 136) {
7               nav.addClass("f-nav");
8           } else {
9               nav.removeClass("f-nav");
10          }
11      });
12
13  });
```

Figura 2.2: Exemplo de código JavaScript utilizando jQuery.

Fonte: Autoria própria.

2.4 CSS

CSS, em português, Folhas de Estilo em Cascata, é utilizado como forma de apresentação de documentos escritos com linguagens de marcação, fornecendo estilos para melhor apresentação das informações.

A principal vantagem de se utilizar CSS é o fato de manter separado a camada de apresentação do conteúdo do documento. O CSS é um formato abrangido por todos os

maiores navegadores de mercado, tornando seu uso presente nos dias atuais.

O CSS é regulamentado pela W3C, consórcio que monitora, controla e padroniza a Web. O principal objetivo do consórcio é desenvolver protocolos e diretrizes que garantam seu crescimento de longo prazo.

O conceito de CSS é direto: criar documentos de folhas de estilo para gerar uma aparência para o site. Com este documento, os aspectos relacionados a aparência do site pode ser modificado de forma simples, alterando apenas algumas linhas neste documento. O fato de poder relacionar várias páginas a um único documento de folhas de estilo também é uma vantagem do CSS, visto que o código não precisa ser reescrito para várias páginas serem estilizadas da mesma forma.

Folhas de estilo possibilitam um grande número de formatações de texto, tamanho de fonte, seleção de família para a fonte, espaçamentos, margens e bordas. CSS também possuem um grande número de medidas para controlar os seus seletores, como polegadas, milímetros, *pixels*, entre outros. (MELONI, 2011).

Para ter mais ideia sobre um código CSS, pode-se ver um exemplo na Figura 2.3.

```
29  .tag:hover {
30    top: -2px;
31    margin: 0.5em 0.5em 0.5em 1.4em;
32    background: #f2dff2; }
33
34  .tag:active {
35    top: 1px; }
36
37  .big {
38    font-size: 1.5em; }
39
40  .giant {
41    font-size: 2.5em; }
```

Figura 2.3: Exemplo de código CSS gerado pelo Sass.

Fonte: Autoria própria.

2.5 Considerações do capítulo

HTML é usado neste projeto para que possa ser possível visualizar em uma página o conteúdo de estilização (CSS) nas *tags* HTML gerado pelo componente Sass, utilizando a plataforma Java EE.

CSS será o código final gerado pelo Sass após sua compilação em uma saída Java EE. O motivo de estilizar os textos de marcação utilizando CSS, é devido a saída de código do preprocessador.

3 PREPROCESSADORES CSS

Preprocessadores são programas capazes de analisar um texto e efetuar conversões léxicas no texto recebido. Conversões léxicas são a substituição de uma gramática esperada para uma gramática programada. Em uma visão geral, os preprocessadores facilitam e reduzem problemas básicos no cotidiano de um desenvolvedor, como um número excessivo de linhas na folha de estilos, repetição de valores, excessivo número de prefixos a serem usados e algumas outras repetições comuns.

A utilidade de usar um preprocessador de CSS é o fato de gerar um código completo utilizando um esforço menor. Atualmente, CSS não permite o reaproveitamento inteligente de trechos de código e também não há a possibilidade de utilização de variáveis. Esses pequenos desvios que ocorrem no dia a dia de um desenvolvedor podem ser suprimidos utilizando um preprocessador de CSS.

3.1 Sass

O Sass, em português, Folhas de Estilo Formidavelmente Sintáticas, é uma metalinguagem escrita com base em CSS que permite escrever folhas de estilos mais limpas e estruturadas. Um código Sass, como exemplificado na Figura 3.1, permite a criação de uma folha de estilo melhor administrável e com uma sintaxe mais elegante para o CSS. A Figura 3.2 mostra o código CSS gerado a partir da Figura 3.1.

SCSS SYNTAX

```scss
$font-stack:    Helvetica, sans-serif;
$primary-color: #333;

body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

Figura 3.1: Exemplo de código escrito em Sass.

Fonte: SASS, 2014.

CSS SYNTAX

```css
body {
  font: 100% Helvetica, sans-serif;
  color: #333;
}
```

Figura 3.2: Código CSS gerado a partir do código Sass.

Fonte: SASS, 2014.

O Sass é uma linguagem de *script* que após compilado, gera arquivos em formato CSS. Ele consiste em duas sintaxes: a original, que é escrita especificamente para o Sass interpretar, e a de saída, que é o próprio CSS. A linguagem interpretada (arquivo com extensão ".scss") têm como características a utilização de elementos para reaproveitamento de código e uso de variáveis. O resultado, também considerado o código CSS final, será gerado sintaticamente mais elegante e de mais fácil compreensão e manuseio.

CSS é uma linguagem declarativa, não uma linguagem de programação. Isto faz com que as propriedades de estilos, seus valores e seus atributos ou pseudo-classes

escritos sejam exatamente o que os navegadores vão interpretar e estilizar a página. Em contramão, uma linguagem de programação permite diferentes maneiras de definir uma lógica para a apresentação, e também possuem elementos que facilitam essas maneiras lógicas, como variáveis, funções e métodos. (FRAIN, 2013). A vantagem de se usar Sass, ao invés do CSS, é o aporte das expressões lógicas citadas e também da facilidade que o preprocessador gera ao não precisar ser repetido como uma folha de estilo.

O Sass também permite que um leque grande de editores de textos possibilitem suporte. Alguns editores renomados estão oferecendo suporte, pois veem no Sass como uma alternativa benéfica ao desenvolvedor na etapa de utilização de folhas de estilo.

Também podem ser listadas as vantagens de usar Sass como ferramenta de preprocessamento de CSS, seguem:

- Usar inclusões de arquivos. Ou seja, é possível adicionar um arquivo dentro de um outro arquivo Sass, apenas usando a propriedade *include*;
- Uso de funções. O Sass é passível de uso de funções predefinidas.
- Variáveis. Assim como outro preprocessador de CSS, o Sass tem como uma de suas característica o uso de variáveis para reuso de informações.
- Expressões Matemáticas. O uso de expressões matemáticas no campo do desenvolvimento CSS está cada vez mais presente, visto que a produção de códigos para diversos dispositivos é uma cultura nos tempos presentes. O uso de expressões matemáticas devem facilitar na hora de desenvolver e escrever *layouts* baseado em mais de um dispositivo.

3.2 LESS

LESS é uma linguagem de folha de estilos dinâmica. Assim como o Sass, o LESS possui sua sintaxe própria, que após ser interpretada, gera um arquivo CSS de acordo com o que foi escrito. Uma característica marcante sobre o LESS é a possibilidade de rodar o interpretador tanto na aplicação do cliente quanto no servidor.

Importante salientar que nem todo ó código LESS é um código CSS válido até que este seja processado, porém, todo o código CSS é um código LESS válido. Isto significa que apenas a mudança da extensão do arquivo, impacta na transformação de um código CSS para LESS. (FOSTER, 2013).

3.3 Stylus

Stylus, como exemplifica a Figura 3.3, é uma ferramenta de preprocessamento de simples instalação e manuseio para gerar CSS. Possui as mesmas características dos outros preprocessadores, o qual processa um arquivo de sintaxe própria para geração de um CSS.

```
fonts = helvetica, arial, sans-serif

body {
  padding: 50px;
  font: 14px/1.4 fonts;
}
```

Figura 3.3: Exemplo de código escrito usando Stylus.

Fonte: Stylus, 2014.

O preprocessador Stylus é notável pela sua sintaxe intuitiva para gerar arquivos CSS e também pela sua criação dinâmica de CSS com uma linguagem de programação flexível e dinâmica. Suas principais características são a ausência de ponto e vírgula, abertura e fechamento de chaves, condicionais e os dois pontos para atribuições. (YAPPA, 2013).

3.4 Considerações do capítulo

O Sass foi escolhido como o preprocessador a ser considerado neste projeto, após análises feitas e também pelo material coletado durante pesquisa prévia. Entre outros motivos está a facilidade de compilação dos arquivos e a sua compatibilidade com Ruby, o que torna fácil a comunicação com a plataforma Java EE, utilizando a biblioteca jRuby.

4 PLATAFORMA JAVA EE

Java EE, é uma plataforma de programação para servidores utilizando a linguagem de programação Java. Java EE fornece uma interface de programação na aplicação (API) e um ambiente para o desenvolvimento de softwares multicamadas. A utilização de Java EE neste projeto se dá na necessidade de criar um compilador para Sass em Java EE. Para criar essa saída, códigos fontes são gerados, utilizando a API desta plataforma, e obedecendo os *servlets* que são criados no software. Java EE utiliza como base a linguagem de programação Java, a qual possui uma implementação, exemplificado na Figura 4.1.

```
236      public String showStories() {
237          setCurrentSprint(sprints.getRowData());
238          return "showStories";
239      }
240
241      public String showDashboard() {
242          setCurrentSprint(sprints.getRowData());
243          return "showDashboard";
244      }
245
246      public Sprint getCurrentSprint() {
247          return currentSprint;
248      }
```

Figura 4.1: Exemplo de código fonte utilizando Java EE.

Fonte: Autoria própria.

4.1 *Servlet*

Para ser possível a utilização de métodos de uma API, são utilizados *servlets*

disponibilizados pela aplicação. *servlets* são classes Java que são usadas para estender as funcionalidades de um servidor ou uma API, além disso, podem responder a quaisquer tipos de requisições. Porém, a utilização neste projeto terá foco na transição de informações entre a biblioteca jRuby e a classe Java.

Um *servlet* é similar a uma extensão proprietária do servidor, com a exceção que deve rodar numa aplicação JVM (Java Virtual Machine), por isto, é uma aplicação considerada segura, estável e com ampla capacidade de portabilidade. Ao contrário de *Applets*, *servlets* operam unicamente no domínio do servidor, ou seja, não requerem o suporte de Java no navegador em que é usada a aplicação. (HUNTER; CRAWFORD, 2001).

4.2 Filter

Filters são funcionalidades das aplicações Java para a Web, mais conhecidas como Java EE, capazes de interceptar dinamicamente requisições e respostas, e manipular as informações que são transportadas nessas requisições e respostas.

Filters são considerados essenciais pela gama de funcionalidades em que podem ser aplicados. Atualmente, os programadores estão procurando uma maneira de deixar os códigos de uma maneira modularizada, a fim de evitar a repetição sem necessidade e também de facilitar a refatoração, caso necessária.

Uma das funcionalidades mais poderosas nos Filters, como mostra a Figura 4.2, é a capacidade de aplicação no momento de implantação. Também notável, é a vinculação de Filters específicos para certos *servlets*, também considerando que estes Filters, podem ser definidos somente para a pré-execução de um *servlet*. (IBM PRESS, 2003).

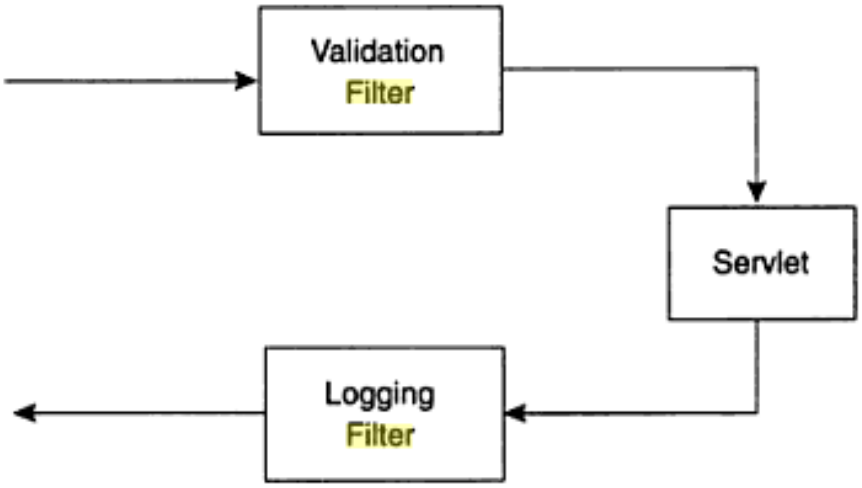


Figura 4.2: Exemplo de fluxo de um Filter.

Fonte: IBM PRESS, 2003.

4.2 Containers Web

Containers Web, mais conhecidos como Servlet Containers, são os componentes de um *web server* que se comunicam com os *servlets* Java. Um Container Web manipula requisições para *servlets*, arquivos JSP (JavaServer Pages) e outras formas de codificação do lado do servidor.

O GlassFish, um Container Web *open-source* mantido pela Oracle Corporation, é uma das referências em implementações de Java EE, com suporte a *servlets* e outras interfaces. Este usa uma derivação do Apache Tomcat, um concorrente mantido pela ASF (Apache Software Foundation), como Container Web para servir um conteúdo web, usando um componente chamado Grizzly, que por sua vez, utiliza Java New I/O para escalabilidade e performance.

A modularidade e a extensibilidade são os diferenciais de como o GlassFish pode se destacar de um simples Container Web, tratando de comandos administrativos para um ambiente de execução mais complexo do que uma simples implantação de arquivos "*.war*" ou "*.jar*". Adicionalmente, o servidor de uma forma intuitiva tem sua partida em poucos segundos, e só é gasto em termos de memória e processamento o tempo de inicialização do servidor, que em média dura cinco segundos. (GONCALVES, 2010).

4.3 Considerações do capítulo

A plataforma Java EE, transmite segurança para o seu usuário, visto que há um leque enorme de API’s, classes, bibliotecas, Containers, e outras aplicações relacionadas com grande popularidade. Este fato, definiu o escopo do projeto no quesito de ambiente de desenvolvimento e implantação do componente gerado.

Outro ponto importante é a facilidade de montar o ambiente para desenvolvimento do componente, visto que o NetBeans, ferramenta escolhida para a edição de códigos, possui o GlassFish instalado por padrão.

5 JRUBY

JRuby é uma implementação da linguagem de programação Ruby implementada em cima da JVM, ou seja, uma aplicação com jRuby implantada, permite a escrita de códigos Ruby em uma aplicação Java. JRuby contém o núcleo completo de classes e sintaxes da linguagem Ruby, assim como as bibliotecas padrão desta.

Apesar de começar a ser implementada em 2001, o interesse em jRuby aumentou consideravelmente nos últimos anos, um reflexo do crescimento expressivo do Ruby, potencializado com a popularidade do Ruby on Rails, um *framework* de desenvolvimento escrito em Ruby.

Para se ter real noção do crescimento do jRuby, a Sun Microsystems, a organização mantenedora do Java e de outras aplicações renomadas de mercado, contribuiu com o sucesso do jRuby desviando funcionários de seu time de desenvolvimento para desenvolver suporte ao jRuby em uma das principais ferramentas de desenvolvimento, o NetBeans. (EDELSON; LIU, 2010).

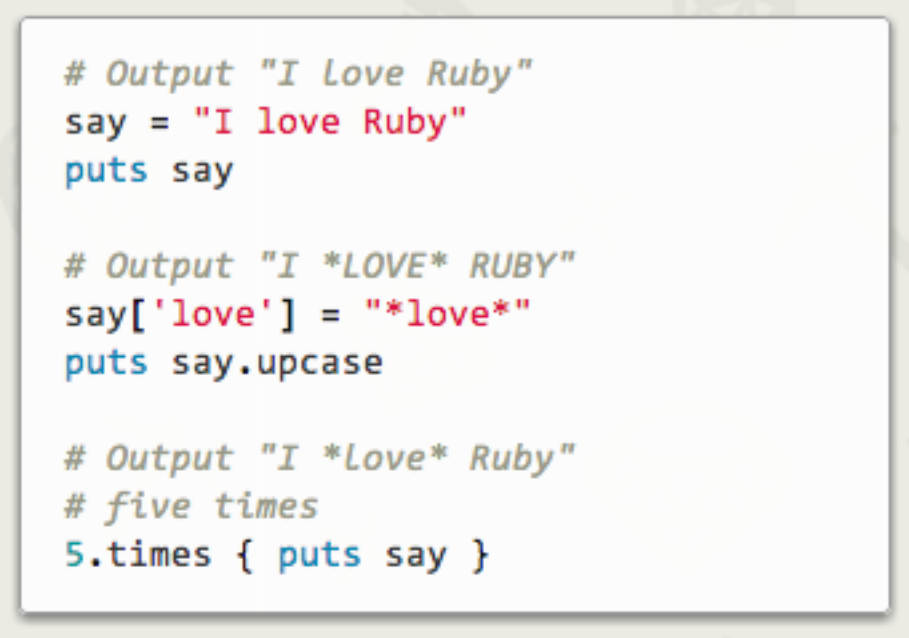
5.1 Ruby

Ruby é uma das linguagens com o maior percentual de crescimento nos últimos anos. Isto se deve porque a mesma se considera a melhor amiga de um programador. E de fato, com a simplicidade e a produtividade que é alcançada com Ruby, sua sintaxe clara e elegante, Ruby se torna uma das linguagens mais apreciadas por todos os desenvolvedores.

Além das características previamente citadas, Ruby é uma linguagem de *script*, orientada a objetos e interpretada. Isto significa que ela é compilada e interpretada em tempo de execução, similar com o que acontece com JavaScript. A vantagem disto é a portabilidade entre múltiplos sistemas operacionais e arquiteturas de *hardware*.

Ruby é uma linguagem puramente orientada a objetos, mas também é usual para estilos de programação procedurais e funcionais. Isto inclui poderosas capacidades de metaprogramação e também podem ser usadas para a criação de linguagens de domínio especifico, do inglês, DSL (*Domain-Specific Language*). (FLANAGAN, 2008).

A Figura 5.1 mostra um exemplo de código em Ruby.



```ruby
# Output "I love Ruby"
say = "I love Ruby"
puts say

# Output "I *LOVE* RUBY"
say['love'] = "*love*"
puts say.upcase

# Output "I *love* Ruby"
# five times
5.times { puts say }
```

Figura 5.1: Exemplo de código em Ruby.

Fonte: RUBY, 2014.

5.2 Considerações do capítulo

Usar jRuby para compilar o Sass utilizando sua biblioteca padrão feita em Ruby é a opção mais completa em termos de componente final para ser disponibilizado para a comunidade.

Com a facilidade de manipular Ruby, e a praticidade do ambiente Java EE, o jRuby fecha com a necessidade da implantação e desenvolvimento previamente analisada.

6 SASS4J

Neste capitulo será descrito como o projeto foi concebido, analisado e desenvolvido. Também serão mostrados os detalhes da análise prévia do projeto, os passos de desenvolvimento, os requisitos funcionais e não funcionais, e os resultados obtidos do desenvolvimento. Todos os quesitos relacionados diretamente a produção do componente serão relatados neste capítulo a fim de trazer de forma transparente a criação deste.

6.1 Concepção

A concepção deste projeto teve-se no momento em que foi percebido que a necessidade de uma ferramenta para auxiliar a usabilidade do preprocessador Sass na plataforma Java EE. Neste momento, foi avaliado se não existia alguma ferramenta no mercado que operasse da mesma maneira. Apesar de ter-se encontrado uma ferramenta que operasse de uma maneira similar, a mesma não tinha sua funcionalidade completa, o que impulsionou ainda mais a criação do sass4j.

Também foi presente na concepção deste projeto a realidade de um desenvolvedor *front-end* que tem seus códigos CSS escritos de maneira repetitiva e de difícil manutenção em projetos de larga escala.

Apesar de tudo, a ferramenta com os mesmos preceitos do sass4j só foi encontrada em meados de maio de 2014, data em que se encontrava em nível avançado de criação do componente sass4j.

6.2 Análise

A análise do projeto foi dividida em subetapas a fim de que o detalhamento do processo seja mais preciso. As subetapas são: análise das necessidades para o

desenvolvimento do projeto, definição do escopo, criação de fluxograma das informações, requisitos funcionais e não funcionais.

6.2.1 Análise das necessidades para o desenvolvimento

O começo da análise do projeto se deu no mapeamento do Servlet Filter da plataforma Java EE. A necessidade de compreender Filter, se teve ao entender que os arquivos CSS/Sass deveriam ser interceptados pela aplicação, a fim de que o componente sass4j tratasse os arquivos interceptados, ou para compilação, ou para obtenção de uma cópia válida que se encontraria na memória do computador.

A segunda parte da análise, deu-se no momento de entender como seria processado o arquivo já interceptado na plataforma. Com esse questionamento, foi levantada a hipótese de criação de uma linguagem de interpretação léxica para compilar corretamente o Sass e transformar em arquivos CSS. Porém, essa opção não contemplaria a solução Sass como um todo no período proposto do projeto.

Sendo assim, a opção a ser tomada foi usar o compilador Sass já existente, escrito em Ruby, em cima de uma aplicação Java EE. Para tal comunicação entre as partes, foi necessário o uso do jRuby, para a passagem de argumentos e comunicação. O jRuby atuou como um comunicador, no momento que foi necessário a criação de uma classe Ruby a qual passava instruções para o compilador Sass.

Após a interceptação do arquivo, e o método de processo do mesmo, foi necessária a definição de como seria armazenado na memória as informações já processadas, com o intuito de poupar requisições que não precisassem ser feitas. Com essa proposta, foi incluído em um mapa as informações necessárias para o processo de armazenamento inteligente do conteúdo processado. Este mapa contém informações como o arquivo CSS original, o arquivo Sass tratado e a data de alteração/criação da versão tratada.

6.2.2 Definição do escopo do projeto

Como decidido anteriormente, a utilização de jRuby como meio de comunicação entre o compilador existente e a plataforma de desenvolvimento, levou a limitar o escopo do projeto com a implementação das funcionalidades já existentes no compilador Sass. Também foi necessário definir o escopo de sintaxes tangíveis de compatibilidade com o sass4j, e com isto, apenas a sintaxe Scss é suportada pela

aplicação.

6.2.3 Diagrama de componentes

O diagrama de componentes do sass4j detalha em como é organizada a classe através da noção de trabalho, ou seja, é utilizado como modelo dos código gerado, auxilia no processo de engenharia reversa e por fim, destaca a função de cada módulo, a fim de reutilizações. Para ilustrar de uma forma clara, a Figura 6.1 traz a imagem que expressa esses conceitos supracitados.

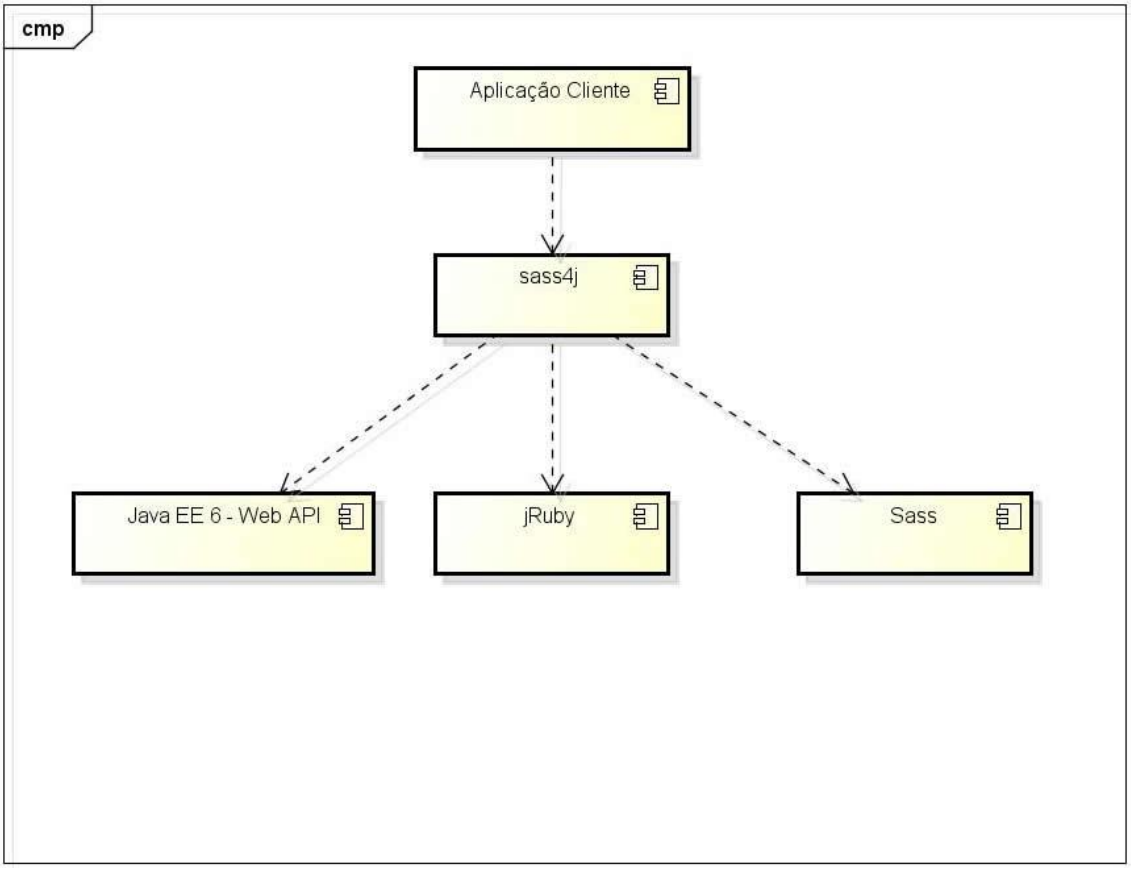


Figura 6.1: Diagrama de componentes da aplicação.

6.2.4 Fluxograma

Para melhor visualização do processo do componente sass4j, foi desenvolvido um fluxograma, o qual pode ser visto na Figura 6.2.

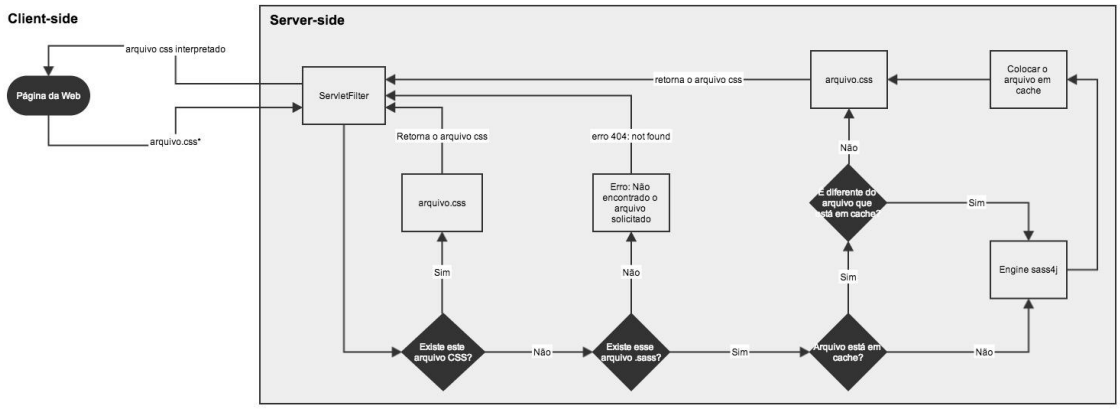


Figura 6.2: Fluxograma da aplicação.

Fonte: Autoria própria.

O começo do processo se dá no carregamento de uma página Web em que o componente sass4j se encontra inserido no projeto. Ao fazer a requisição, é filtrado todos os arquivos CSS existentes nesta requisição, e estes seguem uma série de caminhos. A primeira etapa é a verificação se o arquivo CSS já existe no projeto. Caso já exista, o mesmo é retornado na requisição. Caso não, o próximo passo consiste em verificar se existe o arquivo com o mesmo nome, porém com sua extensão Scss. Caso não exista, é retornado um erro para a aplicação. Caso exista, é verificado se o arquivo se encontra na memória. Caso exista na memória, o mesmo é retornado para a requisição. Caso não se encontre na memória ou o arquivo encontrado seja diferente do arquivo requisitado, o arquivo é reprocessado pelo compilador Sass, colocado e atualizado suas informações na memória, e logo após é devolvido para a requisição.

6.2.5 Requisitos funcionais e não funcionais

Para deixar claro as capacidades do componente, foram definidos requisitos funcionais e não funcionais da aplicação.

6.2.5.1 Requisitos funcionais

Os arquivos tratados pelo componente, deverão estar escritos usando a sintaxe Scss e serão convertidos para arquivos CSS.

6.2.5.2 Requisitos não funcionais

- O componente não converterá o arquivo Scss em tempo real;
- A aplicação deverá prezar pela performance utilizando um mecanismo de armazenamento de informações já processadas;
- Tornar o componente possível de integração com uma aplicação Java EE existente;
- O componente poderá ser incluso em uma aplicação com no mínimo, a versão número seis da aplicação Java EE.

6.3 Implementação

A implementação do sass4j pode ser dividida em três macro etapas. A primeira contém a criação do filtro para interceptação do arquivo, a segunda contém na passagem de informações interceptadas pelo filtro para o componente sass4j interpretar a necessidade e responder de acordo com o caso, e a terceira na implementação do sistema de "cacheamento" de arquivos.

6.3.1 Filtro

Para implementação do filtro, criou-se um *servlet* que implementa a classe *Filter* da plataforma Java EE. Essa implementação gerou três métodos de função do *Filter*, são eles: *init*, onde são processadas as informações no inicio do *Filter*, *doFilter*, onde podem ser manipuladas as informações que são interceptadas pelo *Filter* e por fim, o método *destroy*, o qual é executado ao fim do *Filter*. Para gerar um padrão de interceptação de arquivos, é necessário criar uma chamada do tipo *Annotation*, que pode ser visualizada de uma forma mais clara na Figura 6.3.

```
36  @WebFilter(urlPatterns = "*.css", dispatcherTypes = DispatcherType.REQUEST)
```

Figura 6.3: *Annotation* que padroniza os arquivos que serão filtrados.

Fonte: Autoria própria.

Como pode ser observado, para implementação do sass4j, é estabelecida criada uma padronização de filtragem de todos os arquivos com a extensão ".css". No método *doFilter*, o arquivo Sass é processado e tratado da maneira definida de acordo com o fluxograma. O método *doFilter* ainda é responsável pela devolução do arquivo CSS, se o mesmo já existir na aplicação.

6.3.2 Tratamento de informações dentro do componente

Após a interceptação dos arquivos pelo filtro, os mesmos são verificados se não possuem versão em *cache*, conforme mostrado no diagrama de fluxo. Caso não houver incidência dos mesmos em memória, é feito o processamento do arquivo.

Para processar o arquivo foi criado um método que recebe um *File*, e retorna uma *String* com toda a informação do arquivo. Esse método pode ser visto melhor na Figura 6.4. Na linha 122, o arquivo de entrada entra em uma instrução de repetição, concatenando em uma variável do tipo String todo o conteúdo, até que o fim do arquivo.

```
113     public StringBuilder appendSass(File sassFile) {
114
115         Scanner scanSass = null;
116         try {
117             scanSass = new Scanner(sassFile, "UTF-8");
118         } catch (FileNotFoundException ex) {
119             Logger.getLogger(SassFilter.class.getName()).log(Level.SEVERE, null, ex);
120         }
121         StringBuilder sass = new StringBuilder();
122         while (scanSass.hasNextLine()) {
123             sass.append(scanSass.nextLine());
124         }
125         return sass;
126     }
```

Figura 6.4: Método de criação de *String* para processamento.

Fonte: Autoria própria.

Após unificar o arquivo em apenas uma variável, o arquivo é passado para compilação utilizando a compilador Sass existente em Ruby. Para esse arquivo ser transportado para a camada Ruby, o *servlet* contém um construtor que gerencia e configura a conexão Java EE com a camada Ruby, utilizando jRuby. Para ficar mais claro, pode-se ver na Figura 6.5 como é feita a configuração. Na linha 47, é definido a versão do Ruby que será utilizada. Interessante mostrar também as linhas 52 e 53, onde são definidos os caminhos relativos do projeto. Na linha 57, é feita a execução da

conexão entre a camada Ruby e a camada Java EE.

```
45      public SassFilter() {
46          config = new RubyInstanceConfig();
47          config.setCompatVersion(CompatVersion.RUBY2_0);
48          manager = new ScriptEngineManager();
49          ScriptEngine script = manager.getEngineByName("jruby");
50          engine = (Invocable) script;
51          String rubyFile = SassFilter.class.getResource("../../sass4j.rb").getFile();
52          String sassScript = SassFilter.class.getResource("../../sass.rb").getFile();
53          script.put("sassScript", sassScript);
54          InputStreamReader isr;
55          try {
56              isr = new InputStreamReader(new FileInputStream(rubyFile), "UTF8");
57              script.eval(isr);
58          } catch (FileNotFoundException | UnsupportedEncodingException | ScriptException ex) {
59              throw new RuntimeException(ex);
60          }
61      }
```

Figura 6.5: Construtor com as informações de configuração com a camada Ruby.

Fonte: Autoria própria.

Além do construtor supracitado, foi necessário a criação de um arquivo Ruby (sass4j.rb) que recebe uma variável de configuração de caminho, e também invoca o método de compilação do Sass.

Após ter o jRuby corretamente configurado, é feito a invocação do método de compilação do Sass, por dentro da camada Java EE. Para isto, tem-se um método chamado *compileSass*, o qual recebe um *StringBuilder*, e retorna uma *String*. Esse método pode ser visto com clareza na Figura 6.6. Neste método, na linha 107, está sendo invocado o método que compila o Sass na camada Ruby. Este método tem como retorno uma String com o conteúdo compilado pelo preprocessador.

```
104     public String compileSass(StringBuilder sass) {
105
106         try {
107             return engine.invokeFunction("compile", sass.toString()).toString();
108         } catch (ScriptException | NoSuchMethodException ex) {
109             throw new RuntimeException(ex);
110         }
111     }
```

Figura 6.6: Método o qual invoca o compilador Sass.

Fonte: Autoria própria.

Logo depois, ao retornar uma *String* com o arquivo compilado, o mesmo é inserido na memória, para ter-se uma versão "cacheada", e após, é devolvido para o *Filter* a versão compilada do Sass, ou seja, um arquivo CSS. O método *doFilter* pode ser visto na Figura 6.7.

```
69
70      HttpServletRequest httpReq = (HttpServletRequest) request;
71      HttpServletResponse httpResp = (HttpServletResponse) response;
72      String sassPath = httpReq.getRequestURI().replace("/sass4j", "");
73      File cssFile = new File(httpReq.getServletContext().getRealPath(sassPath));
74
75      if (cssFile.exists()) {
76          chain.doFilter(request, response);
77      } else {
78          File sassFile = new File(httpReq.getServletContext().getRealPath(sassPath).replace(".css", ".scss"));
79          if (sassFile.exists()) {
80              byte[] bytesCss;
81              if(map.containsKey(cssFile.toString()) && map.get(cssFile.toString())[0].equals(String.valueOf(sassFile.lastModified()))) {
82                  bytesCss = map.get(cssFile.toString())[1].getBytes();
83              } else {
84                  StringBuilder sass = appendSass(sassFile);
85                  String sassCompilado = compileSass(sass);
86                  bytesCss = sassCompilado.getBytes();
87                  map.put(cssFile.toString(), new String[]{String.valueOf(sassFile.lastModified()), sassCompilado});
88              }
89              httpResp.setHeader("Content-Type", "text/css");
91              OutputStream os = httpResp.getOutputStream();
92              os.write(bytesCss);
93              os.flush();
94              os.close();
95          } else {
96              httpResp.sendError(404, "SCSS and CSS files not founded.");
97          }
98      }
```

Figura 6.7: Método *doFilter*.

Fonte: Autoria própria.

6.3.3 Otimização de arquivos

Para garantir maior performance, foi implementado um sistema de gerenciamento de arquivos no componente sass4j. Esse gerenciamento, popularmente conhecido como o processo de “cacheamento”, é feito usando um mapa (*WeakHashMap*) de chaves e valores para controlar, consultar e inserir conteúdos. Esse *WeakHashMap* é constituído de um par de *String’s*, sendo que o valor é uma lista de *String’s*, e a chave é constituída de uma *String*. Quando um arquivo é processado pelo componente sass4j, é adicionado na chave o arquivo CSS original, e na lista de valores são adicionadas as seguintes informações: primeiramente a última data de modificação do mesmo e na segunda posição, o arquivo já processado pelo componente.

O *WeakHashMap* é uma implementação da interface de *Map* com uma abordagem diferente. Seu propósito é a remoção automática da chave enquanto ela não estiver sendo utilizada. Sendo mais preciso, a presença de um mapeamento para uma determinada chave, não previne que a mesma não seja descartada pelo coletor de lixo da plataforma Java EE.

Para ficar mais intuitivo, o processo de comparação pode ser simplificado da seguinte maneira: quando é feita a requisição de um arquivo, é comparado se já existe uma versão compilada deste arquivo, se houver, é comparada a última data de modificação, e caso baterem, é devolvido o arquivo já compilado previamente. Entretanto, caso a última data de modificação for diferente, o arquivo é repassado para processamento novamente, e por fim inserido no sistema de "cacheamento".

6.4 Estudo de caso

Como estudo de caso, foi criado uma listagem de etiquetas, estilizadas usando a aplicação sass4j. Para criar a estrutura, foi utilizado um código HTML usando várias âncoras dentro de uma divisória. Essa estrutura pode ser visualizada melhor na Figura 6.8.

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title></title>
5          <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
6          <link href="style.css" rel="stylesheet" type="text/css">
7
8      </head>
9      <body>
10         <div id="content">
11             <a href="#" class="tag giant">sass4j</a><br />
12             <a href="#" class="tag">tiny tags</a>
13             <a href="#" class="tag">that</a><br />
14             <a href="#" class="tag big">scale</a>
15             <a href="#" class="tag">perfectly</a>
16         </div>
17     </body>
18 </html>
```

Figura 6.8: Estrutura HTML do estudo de caso.

Fonte: Autoria própria.

Após a criação desta estrutura, foi necessária a criação de um arquivo de folha de estilo escrito utilizando a sintaxe Scss. A estilização permite a criação visual dos elementos nomeados como rótulos, com a criação de puxadores à esquerda, com uma pequena circunferência no centro a esquerda. O exemplo dessa folha de estilo pode ser verificada na Figura 6.9.

```scss
$fontsize: 13px; $bg: #E8B5B6; $tag: #FFFFFF; $taghover: #F2DFF2; $text: #000000; $bg-white: white; $dg: 45deg;

body {
    color: $text;
    margin: 0;
    font-family: Droid Sans, sans-serif;
    font-size: $fontsize;
    word-spacing: 2px;
    text-align: center;
    background: $bg;
}
.tag:hover {top: -2px; margin: 0.5em 0.5em 0.5em 1.4em;background: $taghover;}
.tag:active{top: 1px;}
.big{font-size: 1.5em;}
.giant{font-size: 2.5em;}
#content{margin: 100px auto 0;}

.tag {
    display: inline-block;
    padding: 0.3em 1.6em;
    background-color: $bg-white;
    position: relative;
    margin: 0 0 0 0.9em;
    line-height: 1.17;
    color: black;
    text-decoration: none;
    margin: 0.5em 0.5em 0.5em 1.4em;
}

.tag::before {
    content: '';
    position: absolute;
    height: 1.25em;
    width: 1.25em;
    top: 0;
    left: 0;
    -webkit-transform-origin: top left;
    -moz-transform: rotate($dg);
    -ms-transform: rotate($dg);
    -webkit-transform: rotate($dg);
    -webkit-transform: rotate($dg);
    background: -moz-radial-gradient(circle 1em at 40% 60%, rgba(0, 0, 0, 0) 0.19em, $tag 0.2em, $tag);
    background: -webkit-radial-gradient(circle 1em at 40% 60%, rgba(0, 0, 0, 0) 0.19em, $tag 0.2em, $tag);
    background: radial-gradient(circle 1em at 40% 60%, rgba(0, 0, 0, 0) 0.19em, $tag 0.2em, $tag);
    z-index: -1;
}
```

Figura 6.9: Folha de estilo do estudo de caso.

Fonte: Autoria própria.

Para que o componente sass4j opere essa folha de estilo, foi criado um projeto no Netbeans, utilizando Java EE, com a aplicação do sass4j em suas bibliotecas. A estrutura montada dentro do projeto, pode ser visualizada na Figura 6.10.

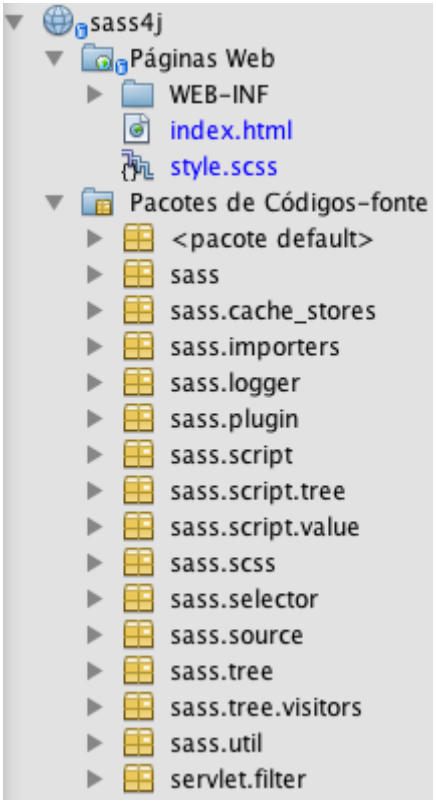


Figura 6.10: Estrutura do projeto em que foi aplicado o sass4j.

Fonte: Autoria própria.

Por fim, foi posta em execução os passos já citados, e o resultado final pode ser visualizado na Figura 6.11.

Figura 6.11: Resultado final da aplicação HTML e Sass utilizando o componente sass4j.

Fonte: Autoria própria.

6.5 Soluções existentes

Como citado anteriormente, foram encontradas algumas soluções com o mesmo conceito do sass4j. Apesar destas soluções partirem do mesmo principio, não foram encontradas soluções que funcionavam perfeitamente como compilador ou componente Sass para a plataforma Java EE. Seguem nos próximos itens soluções com o mesmo principio do sass4j.

6.5.1 JSASS

Jsass é um compilador de Sass utilizando Java. É baseado na utilização de ANTLR (*ANother Tool for Language Recognition*) para a interpretação de expressões em Sass. Apesar das poucas versões de códigos enviados, visto que a última versão tem como data início de 2009, pode-se perceber que é um código de compilação simples, que utiliza de *tokens* para o processo de criação de códigos CSS.

6.5.2 Compass

Assim como o Stylus, LESS e Sass, o Compass também funciona como uma

ferramenta de preprocessamento de CSS, porém sua principal utilização é como um *framework* de CSS escrito em Sass. Suas principais características são: a melhor reusabilidade de padrões, criação facilitada de *sprites*, *mixins* escritos em CSS de uma forma simples e a também fácil criação de extensões das folhas de estilos criadas. Importante salientar que o Compass usa o Sass como o compilador das suas expressões, ou seja, o Compass é uma versão customizada do próprio Sass.

O funcionamento do Compass se baseia em um processo de observação de uma pasta previamente definida. Durante a observação do diretório, ao ser detectada mudanças, uma *trigger* é disparado no processo, e então o Compass entra em ação para recompilar os arquivos, e por fim, gerar um arquivo CSS. (KOSMACZEWSKI, 2013).

6.5.3 Sass-java

Sass-java é um compilador utilizando a sintaxe Sass, em tempo real, utilizando Compass via um s*ervlet* feito em Java EE. Sua funcionalidade é a geração de CSS usando *templates* na sintaxe Sass. Suas principais características são a utilização de Compass para compilar os arquivos Sass e a portabilidade para usar com Maven.

6.6 Resultados obtidos

Por fim do estudo de caso, foi obtido como resultado final um componente de processamento de arquivos Sass (com sintaxe Scss) com capacidade de atuação na plataforma Java EE. Esse componente, sass4j, atingiu sua meta como descrito nos objetivos, com a capacidade de interceptação usando filtros, portabilidade do compilador escrito em Ruby utilizando jRuby e por fim, o armazenamento inteligente de arquivos usando um sistema de *cache*.

7 CONCLUSÃO

Com o final do projeto, entende-se que a criação de um componente para a utilização de um preprocessador de CSS em uma plataforma que não suporta o mesmo dá um retorno em ganho de tempo de produção do usuário final. Também foi observado a fácil aplicação ao contexto Java EE, visto que a portabilidade da conexão entre a interface Java EE e a camada jRuby funcionaram perfeitamente sem grandes problemas. Essa conexão, entre Java EE e jRuby, apesar de não possuir uma documentação consolidada com todos os recursos disponíveis, não ocasionou problemas maiores no desenvolvimento do projeto.

Outro fato importante de se salientar é a usabilidade do projeto, visto que é de fácil compreensão para quem deseja aplicar o componente em seus desenvolvimentos. A facilidade de transformar Sass em CSS utilizando uma plataforma de desenvolvimento como Java EE é impressionante, também devido a sua agilidade no processo de criação e compilação do CSS.

Também é importante salientar a vantagem de usar o sass4j em nível de desenvolvimento final. O sass4j inclui todas as características de usar um preprocessador de CSS, como já falado. Mas também é importante mostrar que o ganho em tempo de desenvolvimento é grande, visto que o tempo de manutenção e também de desenvolvimento é reduzido com a utilização do componente.

7.1 Trabalhos futuros

Cabe deixar aberto para trabalho futuro, a implementação do sass4j para a sintaxe Sass. Atualmente, a versão do sass4j permite apenas a utilização da sintaxe Scss, a sintaxe padrão do compilador, como sintaxe de entrada da aplicação.

Também para futuras implementações, fica a necessidade de criar uma extensão da

biblioteca Java ".jar" da aplicação. Essa extensão pode ser feita através da utilização de *webfragments* para configurar o componente sass4j em qualquer outra aplicação Java EE.

REFERÊNCIAS

BROWN, K; IBM PRESS. **Enterprise Java Programming with IBM WebSphere**. 2nd ed. Boston: Addison-Wesley, 2003. p. 79-80.

DUCKETT, J. **HTML and CSS: Design and Build Websites**. Hoboken: John Wiley & Sons, 2011. p. 20

EDELSON, J; LIU, H. **JRuby Cookbook**. Sebastopol: O'Reilly, 2008. p. 1.

FLANAGAN, D. **JavaScript: The Definitive Guide**. 6th ed. Sebastopol: O'Reilly, 2011. p. 1

FLANAGAN, D.; MATSUMOTO, Y. **The Ruby Programming Language**. Sebastopol: O'Reilly, 2008. p. 1-2.

FOSTER, J. **CSS for Windows 8 App Development**. New York: Apress, 2013. p. 261.

FRAIN, B. **Sass and Compass for Designers**. Birmingham: Packt Publising, 2013. p. 22.

GONCALVES, A. **Beginning Java EE 6 with GlassFish 3**. 2nd ed. New York: Apress, 2010. p. 35.

GOURLEY, D; TOTTY, B; SAYER, M; AGGARWAL, A; REDDY, S. **HTTP: The Definitive Guide: The Definitive Guide**. Sebastopol: O'Reilly, 2002. p. 3-5

HUNTER, J.; CRAWFORD, W. **Java Servlet Programming**. 2nd ed. Sebastopol: O'Reilly, 2001. p. 5-6.

KOSMACZEWSKI, A. **Sencha Touch 2 Up and Running**. Sebastopol: O'Reilly, 2013.

p. 179.

LIBBY, A. **Instant SASS CSS How-to**. Birmingham: Packt Publising, 2013. p. 1

MELONI, J. **Sams Teach Yourself HTML, CSS, and JavaScript All in One**. Indianapolis: Sams Publishing, 2011. p. 1

YAAPA, H. **Express Web Application Development**. Birmingham: Packt Publising, 2013. p. 34.