

INSTITUTO FEDERAL SUL-RIO-GRANDENSE  
CURSO SUPERIOR EM SISTEMAS PARA INTERNET

## **Harmonic: Gerador de Sites Estáticos**

Fabício da Silva Matté

Prof. Sérgio Luis Rodrigues  
Orientador

Pelotas, agosto de 2016

**Fabício da Silva Matté**

# **Harmonic: Gerador de Sites Estáticos**

Trabalho de Conclusão de Curso para obtenção do título de bacharel em Tecnologia em Sistemas para Internet, do Instituto Federal Sul-rio-grandense.

Orientador: Prof. Sérgio Luis Rodrigues

Pelotas

2016

# SUMÁRIO

<b>LISTA DE FIGURAS</b>	<b>6</b>
<b>LISTA DE TABELAS</b>	<b>7</b>
<b>LISTA DE ABREVIATURAS E SIGLAS</b>	<b>8</b>
<b>RESUMO</b>	<b>9</b>
<b>ABSTRACT</b>	<b>10</b>
<b>1 INTRODUÇÃO</b>	<b>11</b>
1.1 Justificativa	12
1.2 Objetivos	13
1.3 Contexto de Pesquisa	13
<b>2 FUNDAMENTAÇÃO TEÓRICA</b>	<b>14</b>
<b>3 METODOLOGIA DE PESQUISA</b>	<b>16</b>
3.1 Sistemas Semelhantes	16
<b>4 MODELAGEM DO SISTEMA</b>	<b>19</b>
4.1 Diagrama de Sequência	20
4.2 Diagrama de Casos de Uso	20
4.3 Diagrama de Classes	21
<b>5 TECNOLOGIAS E FERRAMENTAS UTILIZADAS</b>	<b>26</b>
5.1 JavaScript	26
5.2 Node.js	26
5.3 npm	27
5.4 JSON	27

<b>5.5</b>	<b>Markdown</b>	27
<b>5.6</b>	<b>RSS</b>	28
<b>5.7</b>	<b>gulp</b>	28
<b>5.8</b>	<b>Babel</b>	28
<b>5.9</b>	<b>Nunjucks</b>	29
<b>5.10</b>	<b>Mocha</b>	29
<b>6</b>	<b>DESCRIÇÃO DO SISTEMA</b>	30
<b>6.1</b>	<b>Requerimentos do Sistema</b>	30
<b>6.2</b>	<b>Estrutura do Projeto</b>	31
<b>6.3</b>	<b>Módulo de Criação de um Site Estático</b>	31
<b>6.4</b>	<b>Módulo de Configuração do Site Estático</b>	32
<b>6.5</b>	<b>Módulo de Criação de <i>Posts</i></b>	34
6.5.1	Meta-informações e configurações de <i>posts</i>	34
<b>6.6</b>	<b>Módulo de Criação de Páginas</b>	35
<b>6.7</b>	<b>Módulo de Geração do Site Estático</b>	36
<b>6.8</b>	<b>Módulo de Execução do Site Estático</b>	37
<b>6.9</b>	<b>Módulo de Utilização de Temas</b>	37
<b>6.10</b>	<b>Módulo de Ajuda</b>	40
<b>6.11</b>	<b>Caso de Uso de Criação de Site Estático Simplificada</b>	40
<b>7</b>	<b>CONSIDERAÇÕES FINAIS</b>	43
<b>7.1</b>	<b>Resultados Obtidos e Discussões</b>	43
<b>7.2</b>	<b>Trabalhos Futuros</b>	44
7.2.1	Plugins	44
7.2.2	Melhorias no desenvolvimento de temas	44
7.2.3	Interface gráfica	44
	<b>REFERÊNCIAS</b>	45
<b>8</b>	<b>APÊNDICES</b>	48
<b>8.1</b>	<b>Apêndice 1 - Requisitos Funcionais e Não-Funcionais</b>	48
8.1.1	Entendimento de domínio	48

8.1.2	Requisitos funcionais . . . . .	48
8.1.3	Requisitos não-funcionais . . . . .	49
8.1.4	Classificação de requisitos . . . . .	49
8.1.5	Dependências entre requisitos . . . . .	50
8.1.6	Resolução de conflitos . . . . .	50
8.1.7	Atribuição de propriedades . . . . .	50
8.1.8	Validação dos requisitos . . . . .	51
8.1.9	Gerenciamento de requisitos . . . . .	51

# LISTA DE FIGURAS

Figura 4.1	Diagrama de sequência da execução de um site estático do Harmonic.	20
Figura 4.2	Diagrama de casos de uso do Harmonic. . . . .	21
Figura 4.3	Diagrama de classes do Harmonic. . . . .	23
Figura 4.4	Código fonte do método <code>generateFiles</code> da classe <code>Harmonic</code> . . .	24
Figura 6.1	Instalação do Harmonic. . . . .	30
Figura 6.2	Exemplo do comando "harmonic init". . . . .	32
Figura 6.3	Estrutura do site estático. . . . .	32
Figura 6.4	Exemplo de cabeçalho de meta-informações de um <i>post</i> do Harmonic.	34
Figura 6.5	Geração do site estático pelo Harmonic. . . . .	36
Figura 6.6	Exemplo de instalação de um tema do Harmonic. . . . .	38
Figura 6.7	Site estático do Harmonic utilizando o tema JS Rocks. . . . .	39
Figura 6.8	Manual do Harmonic. . . . .	40
Figura 6.9	Criação de um site estático pelo Harmonic. . . . .	41
Figura 6.10	Criação de uma nova postagem pelo Harmonic. . . . .	41
Figura 6.11	Execução do site estático. . . . .	41
Figura 6.12	Site estático gerado pelo Harmonic com o tema padrão. . . . .	42

# LISTA DE TABELAS

Tabela 2.1	Comparação entre sites estáticos e dinâmicos. . . . .	14
Tabela 3.1	Comparação entre sistemas semelhantes. . . . .	18

# LISTA DE ABREVIATURAS E SIGLAS

CLI	Command-line interface
CMS	Content Management System
JSON	JavaScript Object Notation
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
NVM	Node Version Manager
PHP	PHP: Hypertext Preprocessor
RF	Requisito Funcional
RNF	Requisito Não-Funcional
TC39	Technical Committee 39
RSS	Rich Site Summary



# RESUMO

O projeto Harmonic tem por objetivo desenvolver um gerador de sites estáticos, utilizando como base a plataforma Node.js juntamente com os recursos mais recentes da linguagem JavaScript que foram especificados no padrão ECMAScript 2015 (TC39, 2015), o qual foi finalizado e oficializado como padrão da linguagem JavaScript em meados de junho de 2015. O compilador Babel é utilizado para atingir este objetivo, o qual transforma código que utiliza recursos de especificações recentes e futuras do JavaScript em código que pode ser executado nos motores JavaScript atuais.

O Harmonic roda sobre a plataforma Node.js, que é, de forma resumida, um motor JavaScript combinado com servidor Web, que permite execução do mesmo código JavaScript em todas as principais plataformas (Windows, Linux e Mac), sem possuir as restrições de segurança comumente encontradas no ambiente de navegadores. Ou seja, o código JavaScript executado pelo Node.js tem acesso completo ao sistema de arquivos e rede da máquina hospedeira, e esta é uma das principais capacidades do Node.js das quais o Harmonic utiliza para gerar sites estáticos.

**Palavras-chave:** Gerador de Sites Estáticos, Node.js, ECMAScript 2015.

## **ABSTRACT**

The Harmonic project aims to develop a static site generator, using the Node.js platform as its base together with the most recent JavaScript features that have been specified in the ECMAScript 2015 standard (TC39, 2015), which was completed and officially published as the JavaScript language standard in mid-June 2015. The Babel compiler is utilized to achieve this goal, which transforms code that uses features from the most recent and future JavaScript specifications into code that can be run in the current JavaScript engines.

Harmonic runs on the Node.js platform, which is, basically, a JavaScript engine combined with a Web server, which enables the same JavaScript code to run in all the major platforms (Windows, Linux and Mac), having none of the security restrictions often found in the browser environment. That is, the JavaScript code run by Node.js has full access to the host machine’s file system and network, and this is one of the main Node.js capabilities of which Harmonic uses to generate static sites.

**Keywords:** Static Site Generator, Node.js, ECMAScript 2015.

# 1 INTRODUÇÃO

Ao se construir um site depara-se com inúmeros problemas como, por exemplo, como resolver as partes de *layout* que são repetidas em várias páginas do projeto, outro desafio é como entregar o projeto para o cliente: não seria viável realizar o projeto com um conjunto de tecnologias específicas, como por exemplo PHP (*PHP: Hypertext Pre-processor*) e MySQL, para entregar para um cliente que trabalha com um conjunto de tecnologias incompatíveis. Portanto, comumente usa-se uma ferramenta como o `wget` para percorrer o projeto e transformar as páginas em HTML estático. Embora seja possível automatizar este processo, não seria a solução ideal. Seria necessário algo mais eficaz para suprir essa necessidade. A partir desta carência, surgem então ferramentas próprias para a criação de sites estáticos (EIS, 2013).

Na perspectiva de suprir essas funcionalidades surge a proposta desse trabalho denominado Harmonic que é um gerador de sites estáticos. A ideia é que o usuário crie páginas e até mesmo um blog de forma estática, usando apenas Markdown para formatação de textos e posts.

O projeto Harmonic, é um programa capaz de gerar uma estrutura de pastas e *arquivos fonte* onde o usuário pode criar e gerenciar o conteúdo de seu site, além de poder instalar, criar ou personalizar *temas*, que são grupos de arquivos de *template*.

O Harmonic pode ser instalado de forma gratuita pelo gerenciador de pacotes *npm* (NPM, 2016). Os temas do Harmonic geralmente são disponibilizados através de pacotes também distribuídos pelo *npm*, mas o usuário também possui a opção de criar temas privados sem a necessidade de publicá-los no *npm*, assim como não compartilhá-los com o público.

Outra característica importante do projeto Harmonic é que todo o seu desenvolvimento é de forma aberta, tendo o seu *código fonte* acessível em um repositório hospedado no serviço do GitHub (GITHUB, 2016), assim como todas as discussões, reportagens de problemas, tomadas de decisões e governança do projeto ocorre de forma aberta no próprio *issue tracking system* do GitHub, que é utilizado como uma espécie de sistema de gerenciamento do projeto.

O projeto Harmonic está sendo desenvolvido sobre a plataforma *Node.js*, que possibilita a execução de programas escritos na linguagem de programação JavaScript nos principais Sistemas Operacionais (Windows, Linux e Mac). Desta forma, todo projeto é escrito em uma linguagem de programação simples de escrever e contribuir, além de não possuir limitações comumente encontradas no ambiente de um navegador.

O Harmonic conta com uma CLI (*Command-line interface*), ou seja, todos os seus recursos e comandos são acessados através da linha de comando, utilizando programas como o *Command Prompt* no Windows, ou o Terminal no Linux ou no OS X. Isto facilita a implementação de rotinas de automação envolvendo o Harmonic, seguindo o padrão imposto por este gênero de software que sempre conta com uma interface de linha comando. As interfaces de linha de comando também são muito mais leves que interfaces gráficas, pois não alocam recursos, como por exemplo memória e CPU, do sistema operacional para exibir janelas e componentes de interface. Usuários avançados muitas vezes preferem usar interfaces de linha de comando ao invés de interfaces gráficas, pois são mais fácil no sentido em que todas as ações são realizadas com uma linha de texto, invés de clicar através de sequências de menus, submenus, campos de entrada e botões.

Este trabalho é dividido da seguinte forma nos capítulos que compõem esse trabalho: fundamentação teórica, metodologia de pesquisa, modelagem do sistema, tecnologias e ferramentas utilizadas, descrição do sistema e considerações finais. Na seção 1.1, 1.2 e 1.3 deste capítulo é descrito a justificativa, os objetivos e o contexto de pesquisa respectivamente.

## 1.1 Justificativa

A partir de pesquisas e estudos com ferramentas similares, notou-se que muitos dos sistemas semelhantes são difíceis de instalar, não suportam certos sistemas operacionais como Windows, possuem configurações complexas ou são muito complicados de utilizar. Assim, constatou-se a necessidade de um gerador de sites estáticos multiplataforma, eficiente e prático.

Na proposta deste trabalho, entende-se como multiplataforma os sistemas que sejam compatíveis com a mais ampla gama de sistemas operacionais, independente de *hardware*. Bem como, sistemas eficientes são aqueles que utilizam relativamente pouca

memória e concluem o processamento em tempo apropriado. Como sistemas práticos, entende-se que necessitam de poucos comandos cuja sua sintaxe é extremamente simples, desde a criação até a execução de um site.

Um aprofundamento sobre a pesquisa dos sistemas semelhantes é debatido na seção 3.1.

## 1.2 Objetivos

O objetivo principal deste trabalho é explorar a área de geradores de sites estáticos, criando uma nova ferramenta de geração de sites estáticos multiplataforma, eficiente, fácil de instalar, com o mínimo de complexidade para criar e executar um site estático.

Outro objetivo do projeto Harmonic é explorar ao máximo os novos recursos e propostas do ECMAScript, o padrão oficial da linguagem JavaScript. O próprio nome Harmonic é derivado deste objetivo, uma vez que o processo atual de evolução do padrão ECMAScript foi nomeado Harmony pelo criador da linguagem JavaScript, Brendan Eich (EICH, 2008).

## 1.3 Contexto de Pesquisa

O contexto de pesquisa dessa monografia foi iniciado pelos autores Jaydson Gomes (GOMES, 2016) e Átila Fassina (FASSINA, 2013), juntamente com a organização JS Rocks (ROCKS, 2016) (antigamente chamada de ES6 Rocks).

Em 2014, o autor deste trabalho começou a contribuir para os esforços desta equipe dando continuidade ao projeto Harmonic. Atualmente, atuo como um dos líderes do projeto, onde faço parte da direção avaliando contribuições e conduzindo o projeto.

O Harmonic é um projeto de código fonte aberto disponível no GitHub. Além de contribuições de membros da organização interna JS Rocks, o projeto também aceita contribuições de pessoas e organizações externas as quais serão avaliadas pelos mantenedores oficiais do projeto.

## 2 FUNDAMENTAÇÃO TEÓRICA

A proposta dos sites estáticos é um tema bem popular atualmente, tendo mais de 435 geradores de sites estáticos já existentes e que vem aumentando a cada ano, conforme *Static Site Generators* (STATIC SITE GENERATORS, 2016).

Ao contrário dos sites dinâmicos, que requisitam banco de dados e executam lógica no servidor para cada requisição, todo o conteúdo de um site estático é gerado instantaneamente antes de qualquer acesso por parte de usuários, assim podendo ser hospedado em um servidor que não necessita de banco de dados nem suporte a nenhuma linguagem de programação.

Além disto, como diz David Walsh (WALSH, 2015), sites estáticos são muito mais rápidos que os dinâmicos, pois todo processamento de conteúdo já foi realizado no momento da geração do site estático e não a cada requisição como nos sites dinâmicos. Outro ponto é a segurança: sites estáticos são mais seguros que os dinâmicos pois não possuem lógica de programação no servidor, assim eliminando falhas na programação no lado do servidor.

Foi então realizado um estudo visando as principais características e diferenças entre sites estáticos e dinâmicos, culminando na tabela comparativa 2.1.

Tabela 2.1: Comparação entre sites estáticos e dinâmicos.

	<b>Sites estáticos</b>	<b>Sites dinâmicos</b>
Páginas são servidas diretamente do disco para a rede	Sim	Não
Conseguem buscar páginas inteiramente do cache do navegador	Sim	Não
Necessitam de banco de dados instalado no servidor	Não	Sim
Necessitam de uma linguagem de programação instalada e configurada no servidor	Não	Sim
Executam lógica de programação no servidor, podendo haver falha de segurança e permitindo invasão <i>hacker</i>	Não	Sim
São gerenciáveis	Sim, através de um gerador de sites estáticos	Sim, através de um CMS ( <i>Content Management System</i> )

Fonte: Elaborado pelo autor.

A partir dos dados da tabela 2.1, percebe-se que sites estáticos possuem uma proposta atraente não somente em termos de desempenho e segurança, mas também em termos de custos. Já que sites não necessitam de suporte a qualquer linguagem de programação nem banco de dados no servidor, os custos de hospedagem de sites estáticos são muito mais baixos. Existem até mesmo hospedagens gratuitas e de qualidade que garantem a confiabilidade para os usuários acessarem sites estáticos, como o GitHub Pages (PAGES, 2016), que realmente são completamente gratuitas e não adicionam qualquer tipo de propaganda ao seu site.

Praticamente todos geradores de sites suportam a escrita de conteúdo através da linguagem de marcação Markdown, que é basicamente uma versão simplificada, mais fácil de escrever e ler do que a linguagem de marcação HTML (*HyperText Markup Language*). Markdown é uma ferramenta de conversão de texto para HTML para escritores Web (GRUBER, 2004). O projeto Harmonic também suporta a linguagem de marcação Markdown e o converte para HTML no momento da geração do conteúdo.

O projeto Harmonic é desenvolvido sobre a plataforma Node.js, que suporta todos os principais sistemas operacionais (OS X, Linux, Solaris, FreeBSD, OpenBSD, Windows) o que permite executar o mesmo código em todas estas plataformas. A plataforma Node.js executa código JavaScript, que é uma linguagem de programação leve e bem simplificada (REBERT, 2016), assim facilitando no desenvolvimento. O Node.js também possui mais de 250.000 pacotes públicos publicados no registro de pacotes mais popular, npm (NPM, 2016), o que torna o desenvolvimento muito mais ágil permitindo reutilizar "blocos de construção" para construir sistemas mais elaborados (SORHUS, 2015).

## 3 METODOLOGIA DE PESQUISA

A ideia do projeto surgiu para criar uma nova forma diferente de criar sites, fora do usual que são os sites dinâmicos, os quais possuem banco de dados, programação no servidor.

Através da participação na comunidade JavaScript, surgiu a ideia de criar um gerador de sites estáticos, que oferecesse uma forma mais simples, eficiente e prática de criar sites estáticos.

### 3.1 Sistemas Semelhantes

Foram examinados diversos sistemas semelhantes no início deste projeto, e assim foi constatada a falta de um sistema compatível com todos os principais sistemas operacionais, fácil de instalar e prático que não necessite da instalação de *plugins* para funcionalidades básicas que um usuário normalmente espera de um gerador de sites estáticos.

Foram avaliados os seguintes projetos: Jekyll (JEKYLL, 2016), Metalsmith (SEGMENT, 2016), Hugo (FRANCIA, 2016), Hexo (CHEN, 2016). Nessa avaliação foram observadas várias características como: sistemas operacionais suportados, em qual linguagem de programação foi desenvolvido, que recursos são oferecidos aos usuários, se atualiza o navegador automaticamente ao recompilar o site estático, e como última característica se o código fonte é aberto.

Conforme pode ser observado na tabela 3.1, alguns trabalhos semelhantes possuem características em comum, contudo com a pesquisa destes foi possível conhecer um pouco sobre a área de gerador de sites estáticos, seus objetivos e desafios a serem superados. O estudo desses sistemas semelhantes foi fundamental para elaboração desta monografia.

Note que o Jekyll não suporta o sistema operacional Windows oficialmente, portanto usuários de Windows enfrentam uma grande dificuldade para utilizá-lo. Já o Metalsmith é multi-plataforma e extremamente poderoso, mas não oferece funcionalidades caso o usuário não instale e configure os *plugins* desejados, o que o torna desnecessariamente



complexo para os casos de uso mais simples.

O principal diferencial do Harmonic é sua praticidade. O Harmonic pode ser facilmente instalado em qualquer um dos principais sistemas operacionais, o usuário precisa de apenas dois comandos para criar e executar um site estático, e o Harmonic já conta com todos os recursos essenciais para o gerenciamento do site estático sem necessitar de configurações extras nem instalações de *plugins*.

Tabela 3.1: Comparação entre sistemas semelhantes.

Projeto	Sistemas Operacionais	Escrito em	Recursos	Recompila automaticamente	Atualiza o navegador automaticamente ao recompilar	<i>plugins</i>	Código fonte aberto
Jekyll	Linux, Unix e OS X. Windows não é suportado oficialmente.	Ruby	Posts, páginas, categorias, permalinks, templates personalizados. Suporta os formatos Markdown, Textile e Liquid.	Sim	Não	Sim	Sim
Metalsmith	Suporta praticamente todas plataformas, embora alguns <i>plugins</i> possam não ter suporte a todas as plataformas.	Node.js	Serve como uma abstração para manipulação de arquivos e diretórios, logo não suporta nenhum recurso nativamente e todas as funcionalidades se dão através de <i>plugins</i> .	Sim, através de <i>plugins</i>	Sim, através de <i>plugins</i>	Sim	Sim
Hugo	Linux, OS X e Windows.	Go	Páginas, templates personalizados, seções, ordenação de conteúdo, permalinks, pretty URLs, aliases (redirecionamentos), geração automática e personalizada de sumários, shortcodes, tempo de leitura, contagem de palavras. Suporta o formato Markdown além de outros formatos através de <i>plugins</i> .	Sim	Sim	Sim	Sim
Hexo	Linux, OS X e Windows.	Node.js	Posts, páginas, rascunhos, templates personalizados, permalinks, helpers, internacionalização, integração de serviços de deploy. Suporta o formato Markdown.	Sim	Sim, através de <i>plugins</i>	Sim	Sim
Harmonic	Linux, OS X e Windows.	Node.js	Posts, páginas, rascunhos, templates personalizados, permalinks, internacionalização. Suporta o formato Markdown.	Sim	Sim	Trabalho futuro	Sim

Fonte: Elaborado pelo autor.

## 4 MODELAGEM DO SISTEMA

Após especificar o que o sistema poderia oferecer ao seus usuários, passou-se à etapa de criação dos diagramas para identificar como deve ser o processo de comunicação entre os vários componentes do sistema e a utilização do UML facilitou a modelagem do software.

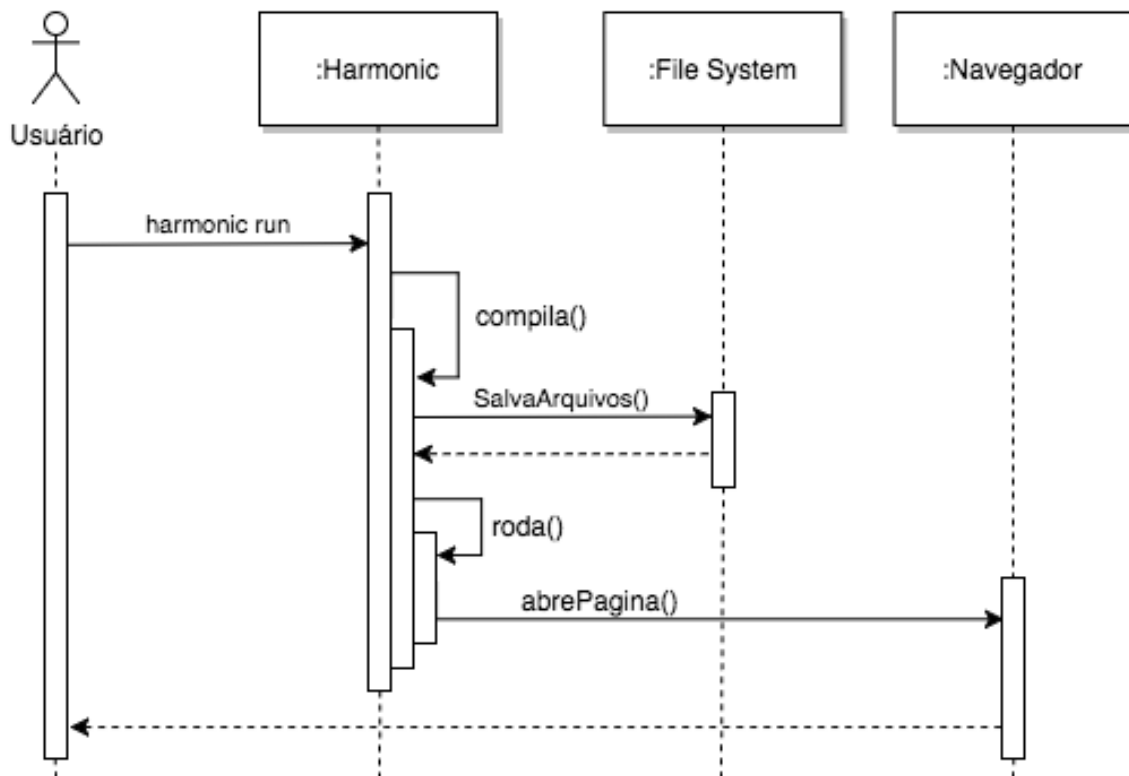
UML é uma linguagem de modelagem que auxilia os desenvolvedores na montagem dos requisitos e do comportamento dos processos no sistema, também atua na descoberta de possíveis necessidades físicas que possam surgir na implementação de uma determinada ferramenta (GUEDES, 2011).

UML é uma metodologia que disponibiliza diversas maneiras para analisar uma determinada questão e neste projeto será utilizado uma abordagem orientada a objetos, que auxilia na reutilização de métodos ou atributos, melhorando o desenvolvimento e a manutenção do sistema. A seção 4.1 apresenta um diagrama de sequência, a seção 4.2 apresenta o diagrama de casos de uso, e a seção 4.3 apresenta o diagrama de classes.

## 4.1 Diagrama de Sequência

Foram criados diagramas de sequência pois estes ajudam a planejar o fluxo de informações e operações a serem realizadas pelas rotinas do sistema. O diagrama de sequência da execução do site estático, que é uma das principais funcionalidades do Harmonic, pode ser conferido na figura 4.1.

Figura 4.1: Diagrama de sequência da execução de um site estático do Harmonic.

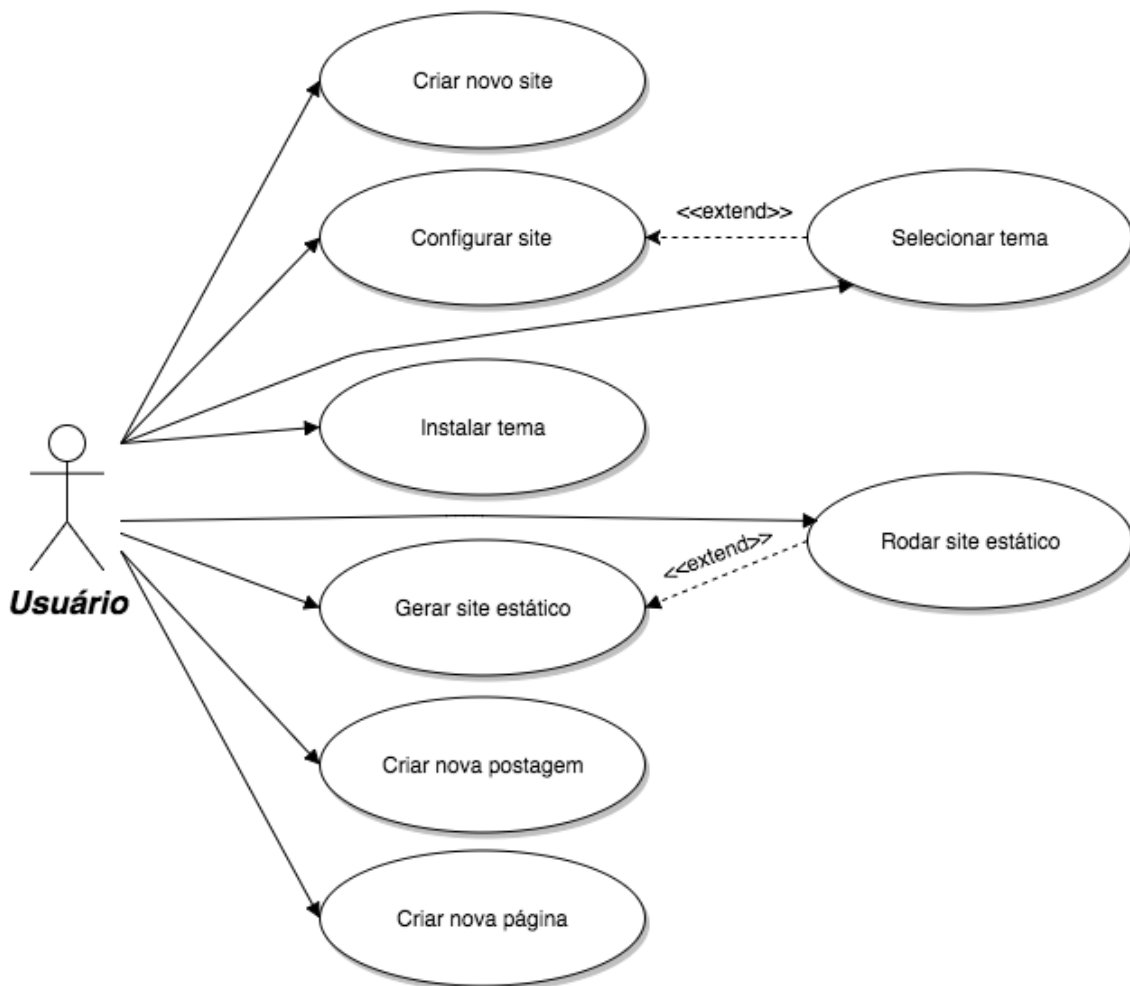


Fonte: Elaborado pelo autor.

## 4.2 Diagrama de Casos de Uso

Foi criado um diagrama de casos de uso pois este facilita mapear as funcionalidades que o sistema deve desempenhar assim como as relações entre elas. O diagrama de casos de uso do Harmonic pode ser conferido na figura 4.2.

Figura 4.2: Diagrama de casos de uso do Harmonic.



Fonte: Elaborado pelo autor.

### 4.3 Diagrama de Classes

Segundo Guedes, o diagrama de classes define a estrutura das classes utilizadas pelo sistema, determinando seus atributos e métodos, e também estabelece as relações e trocas de informações entre as classes do sistema (GUEDES, 2011).

Assim sendo, foi elaborado um diagrama de classes que exibe as principais classes do Harmonic e a comunicação entre elas. Entre elas estão a classe *Harmonic* que encapsula todas as funcionalidades principais do projeto Harmonic, a classe *Helper* que possui várias funções utilitárias utilizadas pela classe *Harmonic*, a classe *Theme* que encapsula o manuseio de arquivos de um determinado tema, a classe *MissingFileError* que é uma especialização da classe nativa *Error* e serve para comunicar um erro devido a

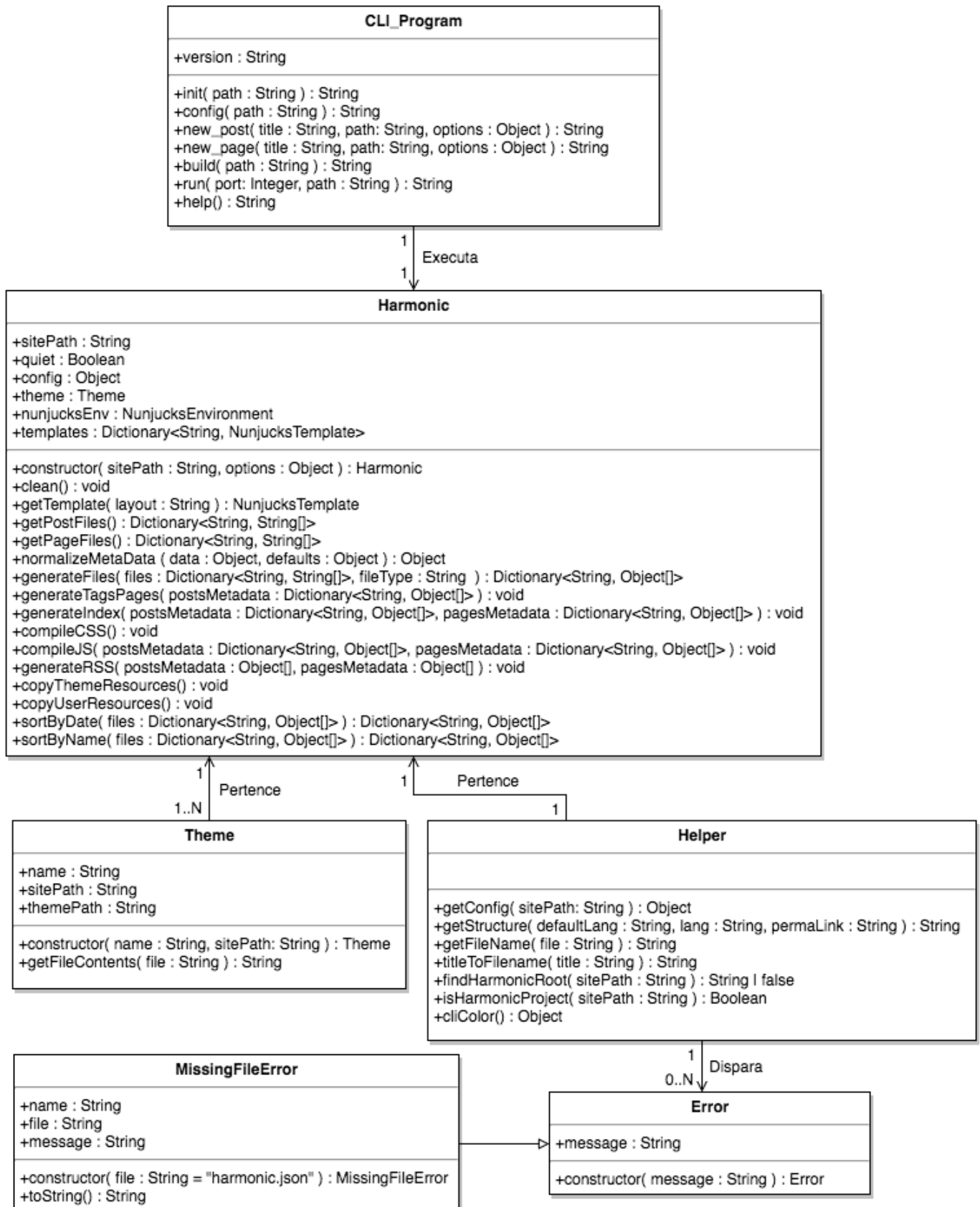
ausência de arquivos essenciais durante uma dada operação, como por exemplo a falta do arquivo de configuração `harmonic.json` caso o mesmo tenha sido excluído ou caso não existe um site estático do Harmonic onde o comando foi executado, e também a classe *CLI\_Program* que representa todas as possíveis interações com a interface de linha de comando do Harmonic. O diagrama de classes do Harmonic pode ser visto na figura 4.3.

Um dos principais métodos da classe Harmonic é o `generateFiles`, cujo código fonte pode ser visto na figura 4.4. O método `generateFiles` da classe Harmonic é responsável pelo processamento e geração dos arquivos de *posts* e páginas do site estático. Este método recebe dois parâmetros como entrada, `files` e `fileType`. O parâmetro `fileType` determina o tipo de conteúdo a ser gerado, sendo ele `post` para *posts* ou `page` para páginas. O parâmetro `files` é um objeto cujas propriedades são as linguagens suportadas pelo site estático e seus valores são os caminhos para os arquivos fontes do tipo de conteúdo a ser gerado na respectiva linguagem.

Este método realiza o *parsing* do cabeçalho de meta-informações do conteúdo (*post* ou página), trata para não publicar conteúdos ocultos nem conteúdos agendados para o futuro, processa o conteúdo Markdown transformando-o em HTML, combina-o com o *template* do tema ativo e escreve os arquivos HTML gerados no caminho de saída, como especificado no arquivo de configuração `harmonic.json`. Então, o método retorna informações sobre os conteúdos gerados de forma propriamente ordenada para que estes sejam utilizados na geração da página inicial e das listagens de *posts* por categoria.

Este método também demonstra o objetivo do Harmonic de explorar os novos recursos e propostas da linguagem JavaScript. O método em si é uma *Async Function*, um novo recurso da linguagem que facilita o controle de fluxo de execução assíncrono utilizando a palavra-chave `await`. As *Async Functions* foram aceitas como parte do padrão ECMAScript em julho de 2016, já sendo incorporadas ao último rascunho da especificação e esperadas a serem publicadas como parte da próxima edição oficial da especificação, o ECMAScript 2017 (HARBAND, 2016). O método também são utilizados recursos experimentais que ainda estão tramitando no processo de desenvolvimento, como a proposta *Function Bind Syntax* (utilizando o operador `::`), além de várias outras funcionalidades padronizadas recentemente, como constantes, *Arrow Functions* e *Template Literals*, que foram padronizadas na edição ECMAScript 2015.

Figura 4.3: Diagrama de classes do Harmonic.



Fonte: Elaborado pelo autor.

Figura 4.4: Código fonte do método generateFiles da classe Harmonic.

---

```

async generateFiles(files, fileType) {
  const langs = Object.keys(files);
  const config = this.config;
  const generatedFiles = {};
  const currentDate = new Date();
  const tokens = config.header_tokens || ['<!--', '-->'];
  const metadataDefaults = {
    layout: fileType
  };

  const filePath = fileType === 'post' ? postspath :
    pagespath;

  await Promise.all([].concat(...langs.map((lang) =>
    files[lang].map(async (file) => {
      const md = new MkMeta(path.join(this.sitePath, filePath,
        lang, file));
      md.defineTokens(tokens[0], tokens[1]);

      const metadata = this.normalizeMetaData(md.metadata(),
        metadataDefaults);
      metadata.content = new
        nunjucks.runtime.SafeString(md.markdown({
          crop: '<!--more-->'
        })));

      const template = this.getTemplate(metadata.layout);
      const filename = getFileName(file);
      const permalink = fileType === 'post' ?
        config.posts_permalink : config.pages_permalink;

      const filePath = permalinks({
        replacements: [{
          pattern: ':year',
          replacement: metadata.date.getFullYear()
        }, {
          pattern: ':month',
          replacement: (metadata.date.getMonth() +
            1)::padStart(2, '0')
        }, {
          pattern: ':title',
          replacement: filename
        }, {
          pattern: ':language',
          replacement: lang
        }
      ]),
      structure: getStructure(config.il8n.default, lang,
        permalink)
    }));

```

---



---

```
    metadata.file = filePath + file;
    metadata.filename = filename;
    metadata.link = filePath;
    metadata.lang = lang;

    const contentHTMLFile = template
      .render({
        [fileType]: {
          content: new
            nunjucks.runtime.SafeString(md.markdown()),
          metadata
        },
        config,
        lang
      })
      .replace(/<!--[\s\S]*?-->/g, '');

    if(fileType === 'page') {
      metadata.content = new
        nunjucks.runtime.SafeString(contentHTMLFile);
    }

    if (metadata.published && metadata.published === 'false')
    {
      return;
    }

    if (metadata.date && metadata.date > currentDate) {
      console.log(clc.info(`Skipping future ${fileType}
        ${metadata.filename}`));
      return;
    }

    const publicFileDirPath = path.join(this.sitePath,
      'public', filePath);
    const publicFilePath = path.join(publicFileDirPath,
      'index.html');
    await mkdirpAsync(publicFileDirPath);

    await fs.writeFileAsync(publicFilePath, contentHTMLFile);
    console.log(clc.info(`Successfully generated ${fileType}
      ${filePath}`));

    generatedFiles[lang] = generatedFiles[lang] || [];
    generatedFiles[lang].push(metadata);
  })))));

  return fileType === 'post' ? this.sortByDate(generatedFiles)
    : this.sortByName(generatedFiles);
}
```

---

## 5 TECNOLOGIAS E FERRAMENTAS UTILIZADAS

### 5.1 JavaScript

JavaScript é uma linguagem de programação orientada a objetos multiplataforma. Dentro de um ambiente hospedeiro (por exemplo, um navegador web), JavaScript pode ser conectado aos objetos de seu ambiente para prover controle sobre eles. O padrão oficial da linguagem JavaScript é o ECMAScript (REBERT, 2016).

O JavaScript obteve uma evolução significativa nos últimos anos, tornando-se assim uma linguagem de propósito geral muito utilizada no mundo. Essa por sua vez é mais conhecida como a linguagem incorporada aos navegadores Web, mas também houve um grande crescimento em sua adoção por parte de servidores e aplicações embarcadas (TC39, 2015).

O projeto Harmonic escolheu esta linguagem de programação devido a sua ampla aceitação no mercado e rápida evolução, além de esta linguagem ser relativamente simples e fácil de trabalhar, contendo centenas de milhares de pacotes comunitários que facilitam no desenvolvimento de praticamente qualquer sistema.

### 5.2 Node.js

Node.js é uma plataforma de código fonte aberto que permite construir aplicações em rede utilizando a linguagem JavaScript. Node.js é construído em cima do V8, uma máquina virtual JavaScript moderna que também é utilizada pelo navegador Web Google Chrome (TEIXEIRA, 2013).

Node.js é um *runtime* JavaScript assíncrono orientado a eventos (ABOUT | NODE.JS, 2016). Isto quer dizer que o Node.js pode ser considerado uma plataforma que combina um motor JavaScript com acesso a recursos do sistema operacional hospedeiro.

A plataforma Node.js roda diretamente sobre o sistema operacional, com isto é possível executar código JavaScript completamente fora do ambiente de um navegador

Web. Assim sendo, o Node.js não interage com uma página da Web diretamente, mas sim com os recursos do computador hospedeiro. Por exemplo, o Node.js pode receber e realizar requisições HTTP (*Hypertext Transfer Protocol*), ler e escrever no sistema de arquivos, e se comunicar com outros processos, *sockets* e *drivers* do sistema operacional (WEN, 2013).

Todo o projeto Harmonic é construído sobre a plataforma Node.js, que realiza toda lógica de negócios e leitura e escritas ao sistema de arquivos do usuário.

### 5.3 npm

O npm é o gerenciador de pacotes padrão da plataforma Node.js, contando com mais de 250.000 pacotes publicados (NPM, 2016) e provendo mais de um bilhão de instalações de pacotes semanalmente (VOSS, 2016).

O projeto Harmonic escolheu o npm como sua plataforma de distribuição e gerenciador de dependências devido ao seu fácil acesso, altíssima resiliência e ubiquidade na comunidade Node.js.

### 5.4 JSON

JSON (*JavaScript Object Notation*) é um formato de troca de informações eficiente. É fácil de ler e escrever manualmente. É fácil de analisar e gerar por computadores, assim como ler e escrever por desenvolvedores (CROCKFORD, 2013).

O projeto Harmonic escolheu o formato JSON para o armazenamento de configurações, pois este formato facilita a operação dos dados tanto pelo sistema quanto pelo usuário final.

### 5.5 Markdown

Como diz Gruber, o criador desta tecnologia, Markdown é uma ferramenta de conversão de texto para HTML projetada para escritores Web (GRUBER, 2004). Markdown é uma sintaxe de formatação em texto puro.

O projeto Harmonic escolheu a linguagem Markdown por ser uma linguagem sim-

ples e popular, com o intuito de facilitar a autoria de conteúdos.

## 5.6 RSS

RSS (Rich Site Summary) é um formato para entrega de conteúdo Web que é atualizado regularmente. Sites de notícias, blogs, resultados de pesquisa, e todos estes tipos de sites podem ser concebidos no formato de uma lista de conteúdos frequentemente atualizados. O RSS é um formato utilizado para entregar este conteúdo no formato de uma lista de *links* e dados sobre cada *link* (NOTTINGHAM, 2005).

O Harmonic escolheu este formato por ser um padrão aberto bem estabelecido, com o intuito de permitir que usuários possam optar por receber notificações e prévias quando conteúdos novos forem publicados.

## 5.7 gulp

O gulp é uma sistema de *build* escrito em Node.js. Sua principal função é automatizar tarefas repetitivas e aprimorar a rotina de trabalho. O gulp é composto por um sistema de gerenciamento de tarefas (*Orchestrator*), um formato virtual que descreve um arquivo (*Vinyl*), um adaptador de que converte uma determinada fonte (como o sistema de arquivos do computador) para o formato Vinyl, uma interface de linha de comando e um conjunto de regras para garantir a qualidade de seus *plugins* (SCHOFFSTALL, 2014).

O projeto Harmonic escolheu o gulp devido a seu amplo número de *plugins*, grande flexibilidade, simplicidade e desempenho. O Harmonic utiliza o gulp para tarefas de compilação do Babel, verificação de qualidade de código e testes unitários.

## 5.8 Babel

O Babel é um compilador genérico multi-propósito para JavaScript. Um de seus principais usos é compilar os recursos da próxima geração do JavaScript para ser compatível com os interpretadores atuais (KYLE, 2016).

O Harmonic escolheu este compilador JavaScript para prover uma maior compatibilidade com as mais diversas versões do Node.js, desde as versões mais antigas, como

a versão 0.10, até as mais atuais, como a versão 6. Assim tornando o Harmonic mais resiliente às implementações de novos recursos ou a falta dos mesmos nas diferentes versões do Node.js.

## 5.9 Nunjucks

Nunjucks é uma linguagem de *templating* sofisticada devolvida pela Mozilla. Os *templates* do Nunjucks suportam vários recursos, entre eles declarações condicionais, laços de repetição, filtros nativos que trabalham com variáveis, herança de *templates* que permite reutilizá-los estendendo e populando outros *templates*, entre outros recursos.

O Nunjucks é parte fundamental da funcionalidade dos temas do Harmonic, pois este provê pontos de inserção nos *templates* do tema para os conteúdos escritos pelo usuário.

## 5.10 Mocha

O Mocha é um ferramenta de testes unitários, o que permite a automatização de testes de funcionalidades através da escrita de suítes de testes (MOCHA.JS, 2016).

O autor deste trabalho escolheu esta ferramenta para aumentar a produtividade no desenvolvimento e evitar regressões de erros de programação.

## 6 DESCRIÇÃO DO SISTEMA

Neste capítulo será abordado os requerimentos do sistema, assim como a estrutura do projeto completa abordando todos os módulos e detalhando todas suas funcionalidades oferecidas. Finalizando este capítulo, será apresentada uma criação de um site estático simplificado, utilizando os recursos básicos do Harmonic.

### 6.1 Requerimentos do Sistema

Para instalar o Harmonic, primeiro é necessário possuir o Node.js instalado juntamente com o npm. O Node.js pode ser instalado diretamente pelo seu site <<https://nodejs.org>>, ou através de vários gerenciadores de pacotes, como o *Advanced Packaging Tool* (apt-get) das distribuições Ubuntu e Debian do Linux, o *Homebrew* do OS X, o *Chocolatey* do Windows, ou programas dedicados para gerenciamento de versões do Node.js, como o n (HOLOWAYCHUK, 2015) e nvm (Node Version Manager) (CASWELL, 2016).

A maioria dos métodos de instalação do Node.js já instala o gerenciador de pacotes npm automaticamente por padrão. Caso o usuário não possua o npm instalado após instalar o Node.js, ele pode ser instalado separadamente também como exibido na documentação do npm (SCHLUETER, 2011), ou compilado a partir de seu código fonte.

Uma vez que o usuário possua a plataforma Node.js e o gerenciador de pacotes npm instalados corretamente, o Harmonic pode ser instalado pelo gerenciador de pacotes npm, através do comando apresentado na figura 6.1. Este comando é digitado em um programa de linha de comando, como o *Command Prompt* ou *Terminal*.

Figura 6.1: Instalação do Harmonic.

---

```
npm install -g harmonic
```

---

Fonte: Elaborado pelo autor.

Este comando instala o Harmonic de forma global, ou seja, a interface de linha de

comando do Harmonic torna-se disponível em todo sistema.

Note que caso usuário esteja usando o sistema operacional OS X, pode ser necessário consertar as permissões do npm antes de instalar pacotes globais, como é documentado no site do npm (CLARK, 2014).

## 6.2 Estrutura do Projeto

O Harmonic divide-se em vários módulos, sendo eles: criação de um site estático, configuração do site estático, criação de *posts*, criação de páginas, geração do site estático, execução do site estático, utilização de temas, além do módulo de ajuda. Na sequência dessa seção será descrito cada módulo.

Assim como a maioria dos pacotes, bibliotecas e ferramentas escritas sobre a plataforma Node.js, o Harmonic está publicado no registro de pacotes do npm e pode ser instalado através do gerenciador de pacotes que faz parte da instalação padrão do Node.js, ou seja, através da ferramenta de linha de comando do npm.

E, similarmente a outros geradores de sites estáticos, o Harmonic conta uma ferramenta de linha de comando que pode ser utilizada para criar um novo site estático, adicionar conteúdo ao mesmo, compilar o site estático e visualizá-lo no navegador.

## 6.3 Módulo de Criação de um Site Estático

O Harmonic permite a criação de novos sites estáticos através do comando `harmonic init`. Este comando gera uma estrutura padrão de pastas e arquivos, onde são guardados todos arquivos fontes do site estático, como pode ser observado na figura 6.3.

O comando `harmonic init` realiza uma série de questionamentos sobre as configurações e personalizações que o site deve possuir (conforme pode ser conferido na figura 6.2), então escreve estas configurações no arquivo de configuração do site estático `harmonic.json`. O usuário pode futuramente modificar estas configurações utilizando o comando `harmonic config` ou editando o arquivo de configuração `harmonic.json` em qualquer editor de texto, já que o arquivo de configuração é um arquivo de texto no formato *JSON*.





- **title:** título do seu site que será exibido no navegador.
- **subtitle:** subtítulo do site. Pode ser utilizado na forma de um *slogan* por alguns temas.
- **description:** descrição do seu site. Temas podem utilizar esta descrição nas *meta tags* do seu site.
- **domain:** endereço onde seu site será hospedado. Esta configuração será utilizada no gerador de RSS (*Rich Site Summary*), e pode ser utilizada por alguns temas também.
- **theme:** nome do tema selecionado.
- **preprocessor:** Preprocessador CSS padrão a ser utilizado na geração do site estático, caso o tema selecionado não especifique um preprocessador CSS.
- **posts\_permalink:** Formato da URL dos *posts* gerados, por exemplo `:language/:year/:month/:title`. O usuário pode personalizar as URLs dos *posts* utilizando os seguintes parâmetros especiais:
  - `:language:` linguagem em que o *post* foi escrito.
  - `:year:` ano em que o *post* foi escrito.
  - `:month:` mês em que o *post* foi escrito.
  - `:title:` título do *post*.
- **pages\_permalink:** Formato da URL das páginas geradas, por exemplo `:language/pages/:title`. Aceita os mesmos parâmetros especiais dos *posts*.
- **index\_posts:** número de *posts* a serem exibidos na página inicial do site. A forma de paginação depende do tema instalado.
- **i18n:** é um objeto contendo as seguintes propriedades de internacionalização:
  - languages:** um `array` de linguagens em que o site está disponível. Os comandos de criação de *posts* e páginas criarão um arquivo de conteúdo para cada linguagem dentro da estrutura fonte do site.

**default:** linguagem padrão do site. A página raiz do site abrirá nesta linguagem, e as postagens e páginas desta linguagem serão geradas sem o segmento `:language` em suas URLs.

## 6.5 Módulo de Criação de *Posts*

O usuário pode criar novos *posts* utilizando o comando `harmonic new_post "<TÍTULO DO POST>"`, que criará um arquivo *Markdown* dentro da pasta fonte de *posts* de cada linguagem que o site estático suporta.

Cada *post* criado é automaticamente adicionado às listagens de *posts* da página inicial e de cada categoria a qual pertence em ordem cronológica inversa (*posts* mais recentes são exibidos primeiro).

### 6.5.1 Meta-informações e configurações de *posts*

Cada arquivo fonte de *post* possui um cabeçalho de meta-informações, no formato de um comentário HTML contendo pares de chaves e valores, como pode ser visto na figura 6.4.

Figura 6.4: Exemplo de cabeçalho de meta-informações de um *post* do Harmonic.

---

```
<!--
layout: post
title: JavaScript iterables and iterators
date: 2015-09-15T04:06:02.428Z
comments: true
published: true
keywords: iterables, iterators, ES2015
description: Understanding JavaScript ES2015 Iterables and
  Iterators
categories: iterables, iterators, ES2015, articles
authorName: UltCombo
authorLink: https://twitter.com/Ult_Combo
authorDescription: Full Stack developer, ECMAScript
  enthusiast, open source lover.
authorPicture:
  https://s.gravatar.com/avatar/326fba1c2980ce0073f6b212acf71ea0
-->
```

---

Fonte: Elaborado pelo autor.

As meta-informações e configurações de *posts* disponíveis são:

- **layout**: nome do arquivo HTML do tema selecionado a ser utilizado na renderização deste *post*. O valor padrão é `post`.
- **title**: título do *post*.
- **date**: data de publicação deste *post*. Pode ser utilizado para agendar uma publicação futura.
- **comments**: Um valor booleano indicando se o *post* deve permitir comentários ou não. Note que nem todos temas suportam comentários. O valor padrão é `true`.
- **published**: Um valor booleano indicando se o *post* está publicado ou não. Pode ser utilizado para despublicar ou ocultar o *post*. O valor padrão é `true`.
- **keywords**: Palavras-chave descrevendo o *post*.
- **description**: Descrição resumida do *post*.
- **categories**: Lista de categorias a qual o *post* pertence, separadas por vírgulas. Você pode especificar qualquer nome de categoria, se a mesma não existir ela será criada.
- **authorName**: Nome do autor do *post*.
- **authorDescription**: Biografia resumida do autor do *post*.
- **authorPicture**: Endereço URL para a foto de perfil do autor do *post*.

## 6.6 Módulo de Criação de Páginas

O usuário pode criar novas páginas utilizando o comando `harmonic new_page "<TÍTULO DA PÁGINA>"`, que criará um arquivo *Markdown* dentro da pasta fonte de *pages* de cada linguagem que o site estático suporta.

Páginas compartilham praticamente todas configurações e meta-informações dos *posts*, com a diferença que sua configuração de `layout` padrão é `page`, ou seja, páginas por padrão utilizam o *template* de `page` do tema selecionado.

Ao contrário de *posts*, páginas não são adicionadas a qualquer listagem automaticamente. Páginas precisam ser vinculadas manualmente a partir de conteúdos ou do tema ativo.

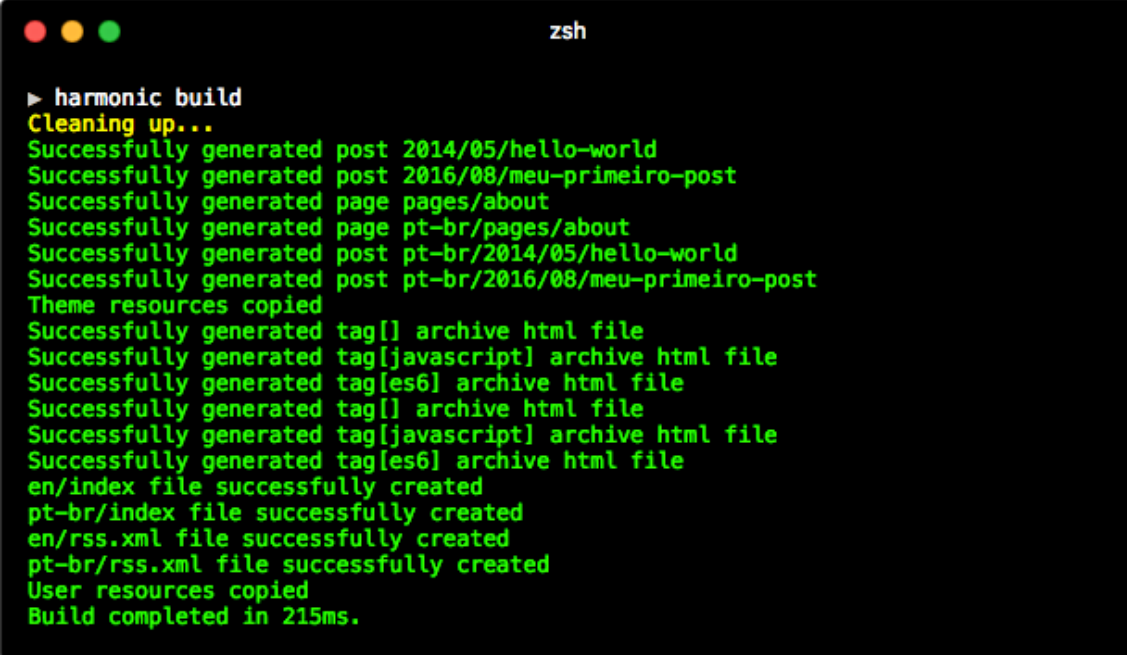
Páginas são ideias para criação de conteúdos personalizados como, por exemplo, informações sobre o site, perfis de autores de conteúdo, galeria de vídeos, ou qualquer outro tipo de conteúdo personalizado.

Vale notar que, embora as páginas e *posts* do Harmonic suportem o formato Markdown, é permitido inserir qualquer tipo de HTML, JavaScript e CSS dentro destes arquivos Markdown, assim provendo um alto nível de personalização para cada conteúdo.

## 6.7 Módulo de Geração do Site Estático

Para consumir o conteúdo do site é necessário primeiro gerar o site estático, que é basicamente um processo de compilação. Para isso, pode-se utilizar o comando `harmonic build`, que gera o site estático a partir dos arquivos fontes (configurações, tema ativo e arquivos Markdown) do seu site Harmonic, assim gerando arquivos HTML, CSS e JS como saída na pasta `public` do site estático, como pode ser visto na figura 6.5.

Figura 6.5: Geração do site estático pelo Harmonic.

A terminal window with a dark background and light green text. The title bar at the top says 'zsh'. The command 'harmonic build' has been executed, resulting in a series of status messages. The output shows the process of cleaning up, generating posts and pages, copying theme resources, generating various tags and files, and finally completing the build in 215ms.

```
zsh

> harmonic build
Cleaning up...
Successfully generated post 2014/05/hello-world
Successfully generated post 2016/08/meu-primeiro-post
Successfully generated page pages/about
Successfully generated page pt-br/pages/about
Successfully generated post pt-br/2014/05/hello-world
Successfully generated post pt-br/2016/08/meu-primeiro-post
Theme resources copied
Successfully generated tag[] archive html file
Successfully generated tag[javascript] archive html file
Successfully generated tag[es6] archive html file
Successfully generated tag[] archive html file
Successfully generated tag[javascript] archive html file
Successfully generated tag[es6] archive html file
en/index file successfully created
pt-br/index file successfully created
en/rss.xml file successfully created
pt-br/rss.xml file successfully created
User resources copied
Build completed in 215ms.
```

Fonte: Elaborado pelo autor.

Com estes arquivos, então, é possível abrir o arquivo `public/index.html` em seu navegador de preferência e navegar pelo site estático.

Note que qualquer edição ou mudança realizada nos arquivos fontes do site, incluindo edição de *posts* e alteração de configurações, requer uma nova compilação para que as mudanças sejam refletidas nos arquivos compilados.

## 6.8 Módulo de Execução do Site Estático

O Harmonic também dispõe do comando `harmonic run`, que além de realizar o mesmo processo de compilação que o `harmonic build`, também inicia um servidor local para servir os arquivos estáticos e automaticamente abre o site estático no navegador padrão do sistema. Este é o método indicado para pre-visualização de seu site estático, pois os navegadores frequentemente bloqueiam vários recursos JavaScript ao acessar os arquivos estáticos diretamente do disco sem um servidor envolvido no processo, o que pode afetar a funcionalidade do site caso você abra os arquivos diretamente no navegador como descrito na seção anterior.

Além disso, o comando `harmonic run` também conta com um recurso de auto-geração de site e auto-recarregamento do navegador ("*live-reload*") sempre que ocorrerem alterações nos arquivos fontes do site, assim proporcionando uma experiência de criação de conteúdos otimizada e muito mais prática e eficiente.

Por padrão, o comando `harmonic run` inicia um servidor local na porta 9356, e abre o site no navegador padrão do sistema automaticamente. É possível customizar estes recursos, passando a porta desejada como primeiro parâmetro do comando (exemplo: `harmonic run 8080`), assim como desabilitar a função de abrir o site automaticamente passando a *flag* `-no-open`.

## 6.9 Módulo de Utilização de Temas

Temas do Harmonic são basicamente um conjunto de arquivos HTML, escritos no sistema de *templating* Nunjucks, seguindo um conjunto de regras que permitem o reuso de arquivos HTML de *template* como estrutura para diferentes *posts*, páginas e listagens com conteúdos diferentes. Temas comumente incluem também arquivos CSS e

JavaScript, assim personalizando ainda mais o estilo e funcionalidades do site gerado.

Por padrão, sites estáticos do Harmonic são criados com o tema `harmonic-theme-default`. O usuário pode instalar ou criar novos temas, e selecionar qual dos temas disponíveis deve ser utilizado na geração do site estático.

Temas do Harmonic podem ser instalados diretamente do registro de pacotes do npm, utilizando o cliente de linha de comando do npm, como pode ser visto na figura 6.6.

Figura 6.6: Exemplo de instalação de um tema do Harmonic.

---

```
npm install harmonic-theme-jsrocks
```

---

Fonte: Elaborado pelo autor.

É recomendável que autores de temas adicionem a palavra-chave `harmonictheme` ao publicar seus temas no registro de pacotes do npm, para que usuários possam encontrá-los mais facilmente utilizando o módulo de busca do npm.

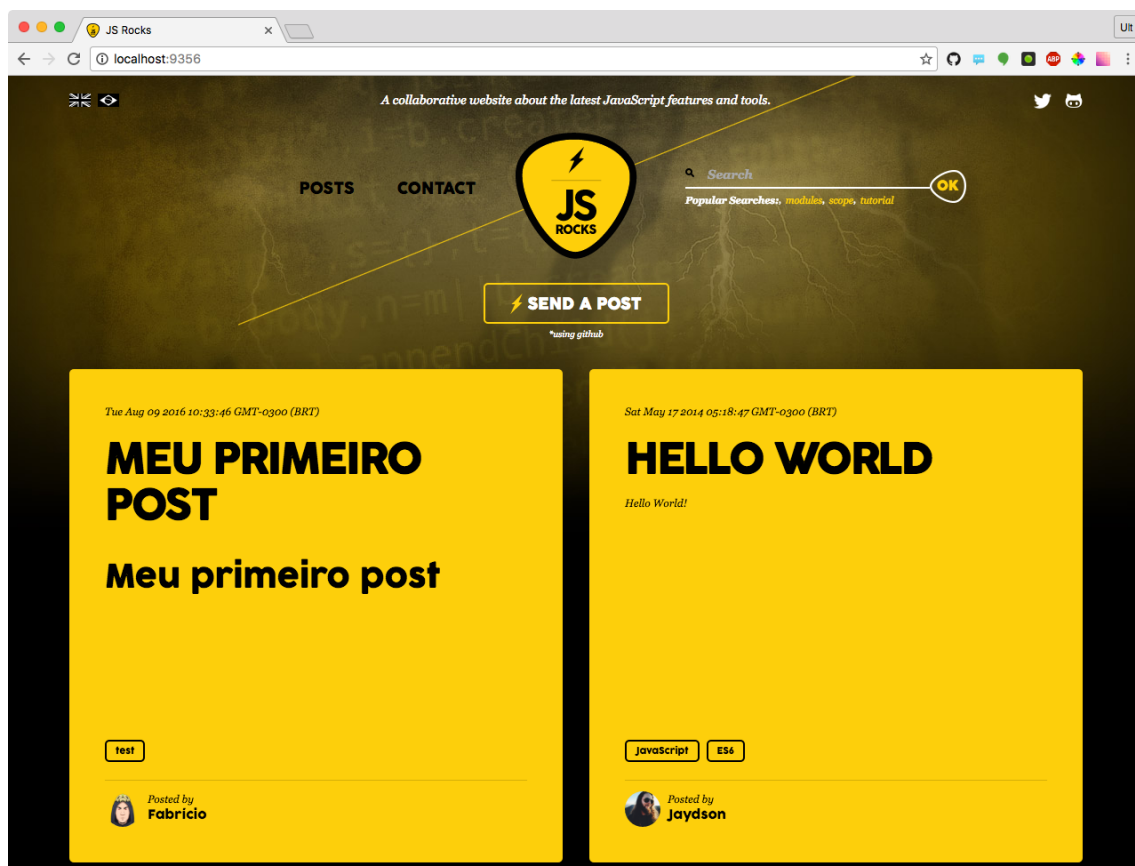
Tendo o tema instalado, o usuário deve selecioná-lo para que este seja utilizado na próxima compilação do site estático. Isto pode ser feito editando o arquivo de configuração `harmonic.json` diretamente em um editor de texto e alterando o valor da propriedade `theme`, ou através do comando `harmonic config`.

Com a troca do tema, o estilo do site pode mudar radicalmente, como pode ser observado na figura 6.7 em comparação com o tema padrão exibido na figura 6.12.

A criação de novos temas geralmente dá-se a partir da edição e personalização de um tema já existente, como o tema padrão ou qualquer outro tema disponível.

Vale notar que, embora o Harmonic e todos seus recursos sejam multiplataforma, compatíveis com todos os principais sistemas operacionais assim como os mais diversos conjuntos de *hardware*, o que define a compatibilidade do site gerado com os navegadores e dispositivos é principalmente o tema escolhido. Os temas mantidos pela organização JS Rocks (`harmonic-theme-default` e `harmonic-theme-jsrocks`) prezam pela responsividade e compatibilidade com o maior número possível de navegadores e dispositivos, mas o mesmo não pode ser garantido para outros temas criados por terceiros.

Figura 6.7: Site estático do Harmonic utilizando o tema JS Rocks.

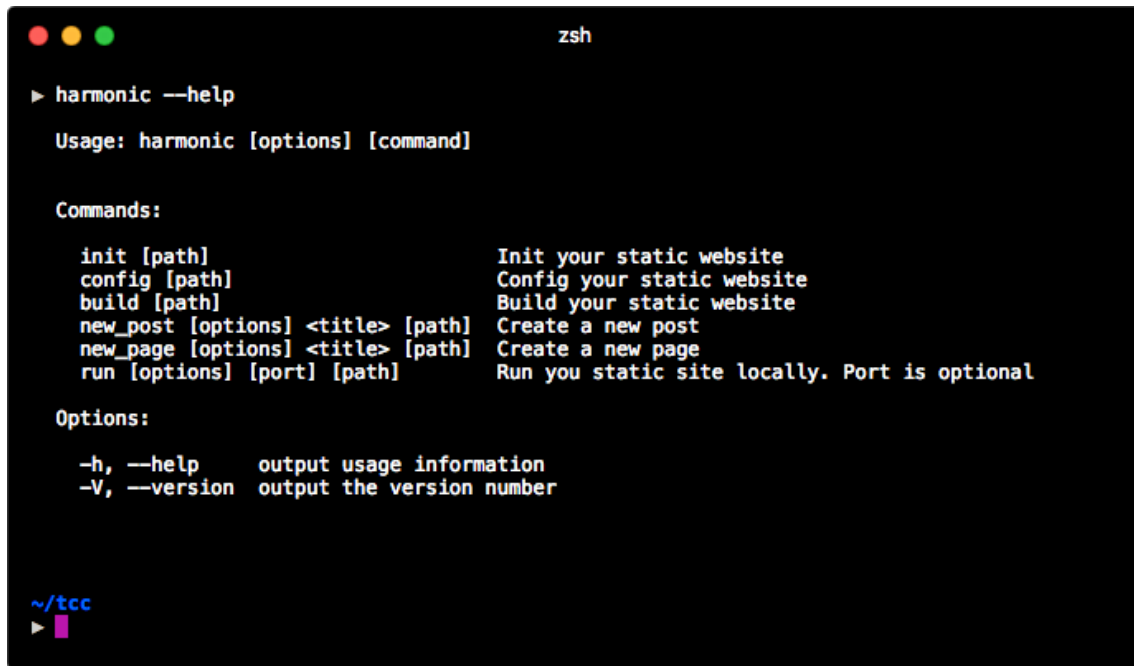


Fonte: Elaborado pelo autor.

## 6.10 Módulo de Ajuda

O Harmonic, assim como praticamente toda ferramenta de linha de comando, conta com um manual do usuário que pode ser acessado através do comando `harmonic -help`, como pode ser visto na figura 6.8.

Figura 6.8: Manual do Harmonic.



```

zsh

▶ harmonic --help

Usage: harmonic [options] [command]

Commands:

  init [path]           Init your static website
  config [path]         Config your static website
  build [path]          Build your static website
  new_post [options] <title> [path] Create a new post
  new_page [options] <title> [path] Create a new page
  run [options] [port] [path] Run you static site locally. Port is optional

Options:

  -h, --help    output usage information
  -V, --version output the version number

~/.tcc
▶

```

Fonte: Elaborado pelo autor.

O Harmonic também conta com uma visão geral de sua documentação em seu repositório no GitHub, a qual pode ser acessada no endereço `<https://github.com/JSRocksHQ/harmonic>`, e a documentação online completa acessível no endereço `<https://github.com/JSRocksHQ/harmonic/tree/master/doc>`.

## 6.11 Caso de Uso de Criação de Site Estático Simplificada

Nesta seção será apresentado uma forma resumida de criação e execução de um site estático usando apenas os recursos principais do Harmonic.

Após ter instalado o Harmonic, o usuário pode criar um novo site estático utilizando o comando apresentado na figura 6.9. Este comando é digitado em um programa



de linha de comando, como o *Command Prompt* ou *Terminal*.

Figura 6.9: Criação de um site estático pelo Harmonic.

---

```
harmonic init
```

---

Fonte: Elaborado pelo autor.

Este comando permite definir a configuração inicial do site estático e gera uma estrutura de arquivos e pastas que servem de base para a construção do site estático. Esta estrutura inicial já conta com uma postagem de exemplo, que pode ser editada ou excluída.

Em seguida, com a estrutura do site estático criada, o usuário pode criar novas postagens através do comando apresentado na figura 6.10.

Figura 6.10: Criação de uma nova postagem pelo Harmonic.

---

```
harmonic new_post "TITULO DA POSTAGEM"
```

---

Fonte: Elaborado pelo autor.

Este comando cria um arquivo Markdown para cada linguagem suportada pelo site estático, dentro da pasta `/src/posts/[LINGUAGEM]`, e automaticamente abre eles para edição no editor de arquivos Markdown (`.md`) padrão configurado no computador do usuário. Estes arquivos armazenam o conteúdo e informações relacionadas à postagem.

Em seguida, o usuário pode gerar o site e visualizá-lo em sua máquina local utilizando o comando exibido na figura 6.11.

Figura 6.11: Execução do site estático.

---

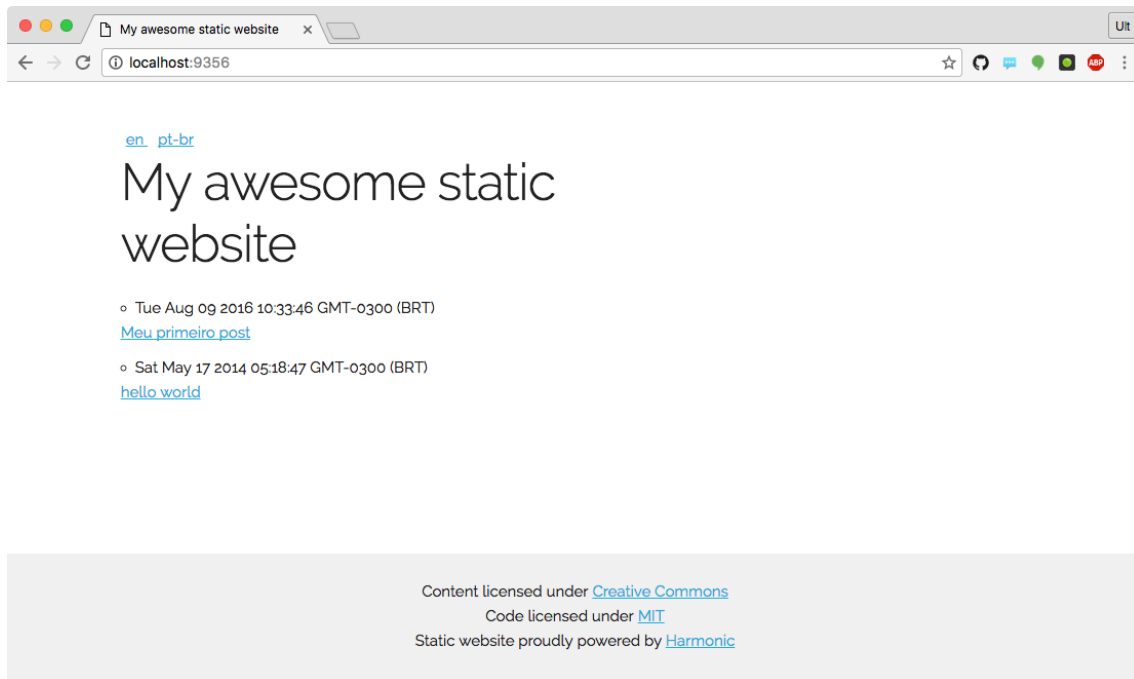
```
harmonic run
```

---

Fonte: Elaborado pelo autor.

Este comando gera os arquivos HTML, CSS e JS a partir do tema selecionado e dos arquivos fontes como *posts* e configurações do site, e então automaticamente abre o site estático no navegador do usuário, como pode ser visto na figura 6.12.

Figura 6.12: Site estático gerado pelo Harmonic com o tema padrão.



Fonte: Elaborado pelo autor.

## 7 CONSIDERAÇÕES FINAIS

Nesse capítulo são resumidos os principais aspectos decorrentes do desenvolvimento deste trabalho. São discutidas os resultados obtidos e discussões, e por fim são apresentados os possíveis trabalhos futuros.

Conforme foi apresentado nesse trabalho sistemas que gerem sites estáticos vem mostrando potencial conforme site Static Site Generators (STATIC SITE GENERATORS, 2016).

O Harmonic é um projeto que pode ser instalado de forma simples e gratuita, através do gerenciador de pacotes npm. Outra característica importante é que todo seu desenvolvimento é de forma aberta e seu código fonte é acessível publicamente.

Outra contribuição importante foi o estudo com sistemas semelhantes, onde foi possível conhecer outros projetos e classificá-los levando em conta suas funcionalidades. Com isto foi possível identificar o diferencial do Harmonic, que é sua praticidade, podendo ser facilmente instalado em qualquer um dos principais sistemas operacionais (Windows, Linux, Mac), sendo assim o usuário com apenas dois comandos é capaz de criar e executar um site estático, além disso ele conta com todos os recursos essenciais para o gerenciamento do site estático sem a necessidade de configurações extras nem instalações de *plugins*.

### 7.1 Resultados Obtidos e Discussões

Como a contribuição deste trabalho, o projeto Harmonic contempla vários propósitos, como por exemplo criar uma ferramenta de geração de sites estáticos multiplataforma, eficiente e prática. Este trabalho atende a este objetivo almejado, já que suporta os mais diversos sistemas operacionais e navegadores, bem como os mais variados *hardwares*.

O outro objetivo que também foi destacado na seção 1.2 era explorar os novos recursos e funcionalidades da linguagem de programação JavaScript. O projeto Harmonic é escrito quase em sua totalidade na linguagem JavaScript, utilizando todos os novos recursos oferecidos por ela.

Como discussão o presente trabalho tem apresentado uma boa aceitação pela co-

munidade, tendo cerca de 150 downloads mensais, com *feedback* na sua grande maioria positivo. Algumas sugestões destes usuários para melhoria deste projeto são debatidas na seção de trabalhos futuros.

## 7.2 Trabalhos Futuros

Dentre os aspectos para continuidade deste trabalho destaca-se:

### 7.2.1 Plugins

Futuramente, desenvolvedores e usuários poderão estender as funcionalidades do Harmonic através de *plugins*. A arquitetura de *plugins* do Harmonic já está sendo discutida e elaborada.

Através de *plugins* será possível adicionar funcionalidades completamente customizadas, como, por exemplo, buscar a foto de perfil do autor a partir de seu email usando serviços de terceiros—como o Gravatar (AUTOMATTIC, 2016)—, compilar arquivos durante o processo de geração do site estático cujos formatos não são suportados nativamente pelo Harmonic, criar um registro de autores para não duplicar os dados do autor nas meta-informações de cada um de seus *posts*, assim como adicionar praticamente qualquer outro tipo de funcionalidade.

### 7.2.2 Melhorias no desenvolvimento de temas

Atualmente, a criação de um tema novo é um processo trabalhoso. A etapa de criação de tema geralmente envolve copiar o tema padrão (ou algum outro tema) e personalizá-lo até que se obtenha o tema desejado.

Estão sendo planejadas melhorias para o desenvolvimento de temas, como novos comandos para a criação de temas e aperfeiçoamento no sistema de recarregamento automático do site para tornar o desenvolvimento de temas claro e eficiente.

### 7.2.3 Interface gráfica

Na continuidade deste trabalho pensa-se em adicionar uma interface gráfica para uma melhor usabilidade para usuários que não estão habituados à interface de linha de comando. Com esta possibilidade, pretende-se atingir um público alvo mais abrangente.

# REFERÊNCIAS

ABOUT | Node.js. Acesso em agosto de 2016, <https://nodejs.org/en/about/>.

AUTOMATTIC. **Gravatar - Globally Recognized Avatars**. Acesso em agosto de 2016, <https://en.gravatar.com/>.

CASWELL, T. **Node Version Manager**. Acesso em agosto de 2016, <https://github.com/creationix/nvm>.

CHEN, T. **Hexo - A fast, simple & powerful blog framework**. Acesso em agosto de 2016, <https://hexo.io/>.

CLARK, L. **npm Documentation - Fixing npm permissions**. Acesso em julho de 2016, <https://docs.npmjs.com/getting-started/fixing-npm-permissions>.

CROCKFORD, D. **The JSON Data Interchange Format**. [S.l.: s.n.], 2013.

EICH, B. **ECMAScript Harmony**. Acesso em agosto de 2016, <https://mail.mozilla.org/pipermail/es-discuss/2008-August/006837.html>.

EIS, D. **Servindo sites estáticos com Jekyll**. Acesso em agosto de 2016, <http://tableless.com.br/jekyll-servindo-sites-estaticos/>.

FASSINA Átila. **Átila Fassina - Front-end Developer**. Acesso em agosto de 2016, <http://atilafassina.com/>.

FRANCIA, S. **Hugo - A Fast & Modern Static Website Engine**. Acesso em agosto de 2016, <https://gohugo.io/>.

GITHUB. **How people build software - GitHub**. Acesso em agosto de 2016, <https://github.com/>.

GOMES, J. **Jaydson Gomes**. Acesso em agosto de 2016, <http://jaydson.org/>.

GRUBER, J. **Markdown**. Acesso em julho de 2016, <https://daringfireball.net/projects/markdown/>.

GUEDES, G. T. A. **UML 2 - Uma Abordagem Prática - 2ª Edição**. Rua Luís Antônio dos Santos 110, 02460-000 – São Paulo, SP – Brasil: Novatec Editora Ltda., 2011.

HARBAND, J. **Finished Proposals**. Acesso em agosto de 2016, <https://github.com/tc39/proposals/blob/master/finished-proposals.md>.

HOLOWAYCHUK, T. **n – Interactively Manage Your Node.js Versions**. Acesso em agosto de 2016, <https://github.com/tj/n>.

JEKYLL. **Jekyll - Simple, blog-aware, static sites - Transform your plain text into static websites and blogs**. Acesso em agosto de 2016, <https://jekyllrb.com/>.

KYLE, J. **Babel User Handbook**. Acesso em agosto de 2016, <https://github.com/thejameskyle/babel-handbook/blob/master/translations/en/user-handbook.md>.

MOCHA.JS. **Mocha - the fun, simple, flexible JavaScript test framework**. Acesso em agosto de 2016, <https://mochajs.org/>.

NOTTINGHAM, M. **RSS Tutorial**. Acesso em agosto de 2016, <https://www.mnot.net/rss/tutorial/>.

NPM. **npm**. Acesso em julho de 2016, <https://www.npmjs.com/>.

PAGES, G. **GitHub Pages - Websites for you and your projects**. Acesso em agosto de 2016, <https://pages.github.com/>.

REBERT, C. **Introduction - JavaScript**. Acesso em agosto de 2016, [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction#What\\_is\\_JavaScript](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction#What_is_JavaScript).

ROCKS, J. **JS Rocks organization**. Acesso em agosto de 2016, <https://github.com/JSRocksHQ>.

SCHLUETER, I. Z. **npm: a package manager for javascript - Fancy Install (Unix)**. Acesso em julho de 2016, <https://github.com/npm/npm#fancy-install-unix>.

SCHOFFSTALL, E. **gulp - The vision, history, and future of the project**. Acesso em agosto de 2016, <https://medium.com/@contrahacks/gulp-3828e8126466>.

SEGMENT. **Metalsmith - An extremely simple, pluggable static site generator**. Acesso em agosto de 2016, <http://www.metalsmith.io/>.

SORHUS, S. **Micro-modules**. Acesso em agosto de 2016, <https://github.com/sindresorhus/ama/issues/10#issuecomment-117766328>.

STATIC Site Generators. Acesso em agosto de 2016, <https://staticsitegenerators.net/>.

TC39. **ECMAScript 2015 Language Specification - ECMA-262 6th Edition**. [S.l.: s.n.], 2015.

TEIXEIRA, P. **Instant Node.js Starter**. Livery Place, 35 Livery Street, Birmingham B3 2PB, UK: Packt Publishing, 2013. 3p. n.cap. 1.

VOSS, L. **The npm Blog - how many npm users are there?** Acesso em julho de 2016, <http://blog.npmjs.org/post/143451680695/how-many-npm-users-are-there>.

WALSH, D. **An Introduction to Static Site Generators**. Acesso em agosto de 2016, <https://davidwalsh.name/introduction-static-site-generators>.

WEN, B. **6 things you should know about Node.js.**  
Acesso em agosto de 2016, <http://www.javaworld.com/article/2079190/scripting-jvm-languages/6-things-you-should-know-about-node-js.html>.

## 8 APÊNDICES

### 8.1 Apêndice 1 - Requisitos Funcionais e Não-Funcionais

Nesta seção é debatido o entendimento do domínio, os requisitos funcionais e não-funcionais, entre outros tópicos de requisitos.

#### 8.1.1 Entendimento de domínio

Gerador de sites estáticos, construído sobre a plataforma Node.js e compatível com todos os principais sistemas operacionais (Windows, Linux, OS X).

#### 8.1.2 Requisitos funcionais

[RF 001] O sistema deve fornecer um comando para inicialização de um novo site estático, que deve criar a estrutura de pastas e arquivos necessária.

[RF 002] O sistema deve fornecer um comando para criação de um novo post, que deve criar um arquivo markdown com o nome desejado na pasta de posts de cada linguagem que o site suporta, contendo meta-informações básicas sobre o mesmo e o título escolhido.

[RF 003] O sistema deve fornecer um comando para criação de uma nova página.

[RF 004] O sistema deve permitir instalar novos temas.

[RF 005] O sistema deve permitir trocar o tema ativo, escolhendo entre os temas instalados.

[RF 006] O sistema deve fornecer um comando para geração do site estático a partir dos arquivos fontes (configurações, tema ativo e arquivos markdown), gerando arquivos HTML, CSS e JS como saída.

[RF 007] O sistema deve fornecer um comando para visualizar o site estático no computador do usuário criando um servidor local.



[RF 008] O sistema deve fornecer a possibilidade de atualizar o navegador automaticamente ao realizar alterações nos arquivos fonte enquanto o servidor local estiver sendo executado.

[RF 009] O sistema deve fornecer um comando de ajuda que explica a funcionalidade e parâmetros de todos os comandos.

[RF 010] O sistema deve fornecer uma forma de criar novos temas.

[RF 011] O sistema deve fornecer uma forma de alterar as configurações de um site já criado.

### **8.1.3 Requisitos não-funcionais**

[RNF 001] O sistema deve manter compatibilidade com todos os principais sistemas operacionais: Windows, Linux e OS X.

[RNF 002] O sistema deve ser desenvolvido sobre a plataforma Node.js, mantendo a compatibilidade com a versão 0.10 e versões mais recentes, como a 0.12, 4, 5 e 6.

[RNF 003] O sistema deve ser distribuído pelo gerenciador de pacotes npm.

### **8.1.4 Classificação de requisitos**

#### **8.1.4.1 Gerenciamento**

- [RF 001]
- [RF 009]
- [RF 011]

#### **8.1.4.2 Edição**

- [RF 002]
- [RF 003]
- [RF 007]
- [RF 008]

#### **8.1.4.3 Personalização**

- [RF 004]
- [RF 005]
- [RF 010]

#### **8.1.4.4 Publicação**

- [RF 006]
- [RF 007]

### **8.1.5 Dependências entre requisitos**

- [RF 005] depende de [RF 004].
- [RF 008] depende de [RF 007].

### **8.1.6 Resolução de conflitos**

Não existem conflitos.

### **8.1.7 Atribuição de propriedades**

#### **8.1.7.1 Essenciais**

- [RF 001]
- [RF 006]
- [RF 011]
- [RNF 001]
- [RNF 003]

#### **8.1.7.2 Importantes**

- [RF 002]
- [RF 003]

- [RF 004]
- [RF 005]
- [RF 007]
- [RF 009]
- [RNF 002]

#### **8.1.7.3 Desejáveis**

- [RF 008]
- [RF 010]

### **8.1.8 Validação dos requisitos**

Será feito através da geração de casos de testes no intuito de desenvolver testes específicos para cada requisito.

### **8.1.9 Gerenciamento de requisitos**

Durante o trabalho não houve alteração de requisitos.