UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

JOSÉ LUIS DAMAREN JUNIOR

Implementando uma Página Dinâmica com um Gerador de Sites Estáticos

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Prof. Dr. Marcelo Soares Pimenta

Porto Alegre
2018

AGRADECIMENTOS

Obrigado

RESUMO

O uso da Internet mudou bastante desde o seu início de sites estáticos, até hoje, com aplicações altamente interativas. Naturalmente, as tecnologias que existiam no início foram substituídas por outras, que permitem funcionalidades como incluir *scripts* nas páginas Web, tornando-as muito mais interativas e versáteis. As páginas Web podem ser classificadas como dinâmicas ou estáticas de acordo com o seu comportamento. Páginas estáticas são, em geral, mais seguras e mais rápidas que páginas dinâmicas, mas são mais limitadas. O surgimento de novas aplicações Web também trouxe mais complexidade. Gerenciar e criar um site com diversas funcionalidades e páginas dinâmicas não é uma tarefa simples. Os sistemas de gerenciamento de conteúdo (CMS) são ferramentas que facilitam a criação e manutenção de um site dinâmico. Os CMS podem ser usados por pessoas que têm pouco ou nenhum conhecimento sobre desenvolvimento Web. Por isso e por implementarem páginas com funcionalidades dinâmicas possuem diversas vulnerabilidades. Apesar das suas limitações, sites estáticos crescem em popularidade atualmente, em função do surgimento de serviços (e.g. sistema de comentários, formulários, integração com redes sociais, serviço de busca) que podem ser integrados ao site sem que ele tenha um lado do servidor e também pela existência dos geradores de sites estáticos (GSE), que são ferramentas que auxiliam na criação e manutenção de um site estático, que pode ser gerado através de um comando a partir de arquivos de *template*, conteúdo e dados. Reimplementou-se uma página Web já existente que possui algumas funcionalidades dinâmicas. A nova implementação da página consiste em uma página com algumas das funcionalidades implementadas de forma estática. A página mantém toda a sua funcionalidade, mas não é mais gerada no servidor para cada requisição e realiza uma requisição a menos durante a sua execução, apresentando vantagens de segurança e desempenho. A página foi implementada com o Jekyll. É possível implementar de forma estática funcionalidades de uma página dinâmica usando um GSE. Isso exige que o desenvolvedor compreenda quais funcionalidades da página podem ser implementadas de forma estática e tenha um entendimento básico de linha de comando para poder usar o GSE, que é uma alternativa a CMS, respeitando os limites de páginas estáticas e as exigências de funcionalidades dinâmicas.

Palavras-chave: Geradores de sites estáticos. Sistemas de gerenciamento de conteúdo. Sites estáticos. Sites dinâmicos. Jekyll.

Implementing a Dynamic Page with a Static Site Generator

ABSTRACT

The Internet use has changed a lot since its beginning with static sites until today, with highly interactive applications. Naturally, the technologies that existed in the beginning were replaced by others, that provide functionalities such as including scripts on Web pages, making them more interactive and versatile. Web pages can be sorted as dynamic or static according to their behavior. Static pages are, in general, faster and safer than dynamic pages, but also more limited. The emergence of new Web applications also brought more complexity. To manage and to create a site with several functionalities and with dynamic pages is not an easy task. Content Management Systems (CMS) are tools that ease the creation and maintenance of a dynamic site. CMS can be used by people with little or without knowledge of Web development. For that, and for implementing pages with dynamic features, they have many vulerabilities. Despite its limitations, static sites grow in popularity nowadays, because of the emergence of services (e.g. comment system, forms, social network integration, search) that can be integrated to the site without it needing a server side and also because of the existence of Static Site Generators (SG), which are tools that help the creation and maintenance of a static site, that can be generated with a command from *template*, content and data files. We reimplement a previously existing Web page that contains some dynamic features. The new implementation consists of a page with some of the features implemented in a static way. The page maintains all its functionalities, but is not generated in the server for every request anymore and performs one less request during its execution, showing security and performance advantages. The page was implemented with Jekyll. It's possible to implement in a static way the functionalities of a dynamic page using an SG. That requires that the developer understand which features can be implemented in a static way and possesses a basic knowledge of the command line for using the SG. SG are an alternative to CMS, respecting the limitations of static pages and the demands of dynamic features.

Keywords: Static Site Generators. Content Management Systems. Static Sites. Dynamic Sites. Jekyll.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CMS	Sistemas de Gerenciamento de Conteúdo
GSE	Geradores de Sites Estáticos
WP	WordPress
FTP	File Transfer Protocol

SUMÁRIO

1 INTRODUÇÃO

No seu início, a Internet era utilizada para disponibilizar grandes quantidades de informação entre, principalmente, pesquisadores de instituições diferentes que, antes da Web, recorriam a práticas mais lentas para se comunicarem. As páginas não ofereciam uma grande quantidade de interação com o usuário, salvo cliques em links que levam de uma página contendo texto a outra página contendo texto.

Naturalmente, ao longo do tempo, as pessoas começaram a se aproveitar do potencial da Internet, criando novos usos, que vão desde e-mail até *e-commerce*. As páginas Web, que no HTML 1 não ofereciam possibilidade de interação e eram extremamente simples, passaram a permitir *scripts* a partir do HTML 4. Atualmente, há diversas tecnologias que são usadas para criar páginas interativas, e essas tecnologias são chamadas de HTML dinâmico (TANENBAUM; WETHERALL, 2011).

As novas aplicações exigiam algo a mais que HTML. Interação com o usuário, comunicação do cliente com o servidor, atualização de conteúdo em tempo real. Dessa forma, as limitações de sites puramente estáticos foram ficando evidentes. Com sites e páginas cada vez mais complexos e com mais funcionalidades, surgiu a necessidade de ferramentas de auxílio de criação e gerência de sites. Linguagens de *template* começaram a surgir, e também ferramentas de desenvolvimento Web como FrontPage e Dreamweaver. Essas ferramentas permitiam gerar um website a partir de *templates*, parciais e até banco de dados SQL[1].

Posteriormente, sistemas como WordPress, Drupal e Joomla apareceram. Eles oferecem a possibilidade de se criar sites com pouco ou até nenhum conhecimento técnico de desenvolvimento Web, com interface gráfica e facilidade de uso. Essas ferramentas permitem, juntamente com as suas extensões, a criação de sites diversos, por usuários muitas vezes mais preocupados com funcionalidade e facilidade de uso do que com segurança[2].

Sem dúvida a Internet mudou, e continua mudando, desde o seu início. Novos usos antes nem pensados surgem, e, naturalmente, a forma como as pessoas utilizam a Internet muda também. Naughton (1999) afirma:

> At first the Net was treated as a kind of low-status weirdo craze, akin perhaps to CB radio and the use of metal detectors. Then it became the agent of Satan, a conduit for pornography, political extremism and subversion. Next it was the Great White Hope of Western capitalism, a magical combination of shopping mall and superhighway which would enable us to make purchases without leaving our recliners. Then it was a cosmic failure because it turned out that

[1] <https://www.smashingmagazine.com/2015/11/modern-static-website-generators-next-big-thing>
[2] <https://www.cmscritic.com/functionality-and-ease-of-use-are-key-factors-in-content-management-system-purchases>

Internet commerce was - shock! horror! - insecure.

Objetivos

Isso traz o questionamento sobre qual a melhor abordagem para uma certa atividade. Mais especificamente, neste trabalho, investiga-se uma das formas de se criar e gerenciar um site, que é com sistemas de gerenciamento de conteúdo. Aponta-se algumas de suas vantagens e desvantagens, e.g., eles possuem uma grande facilidade de uso, mas também têm muitas vulnerabilidades. Além disso, discute-se sobre em que situações e por quais motivos eles podem ser substituídos por geradores de sites estáticos.

Também discute-se aqui sobre páginas Web estáticas e dinâmicas, e os motivos pelos quais certas aplicações, como *e-commerce*, buscas em catálogos de livros ou músicas, navegação em mapas e colaboração em documentos, exigem páginas dinâmicas. Experimentamos com a criação de uma página dinâmica, alterando a página de tal forma a deixá-la mais estática, sem perder funcionalidade, usando o Jekyll[3], um GSE.

O objetivo do trabalho é reimplementar uma página (já existente) que possui comportamento dinâmico, modificando parte do seu comportamento para que seja estático, a fim de aproveitar as vantagens do comportamento estático (segurança e performance). Implementa-se a página com o Jekyll, para utilizar as facilidades que a ferramenta oferece para criação de sites.

Organização do texto

O capítulo 2 apresenta as definições de sites e páginas dinâmicos e estáticos, assim como vantagens e desvantagens de cada. O capítulo 3 mostra uma introdução a CMS e discute sobre os sistemas. O capítulo 4 introduz geradores de sites estáticos e explica o funcionamento do Jekyll em certo detalhe. O capítulo 5 descreve como uma página Web dinâmica já existente foi implementada com o Jekyll, com duas de suas características dinâmicas implementadas de forma estática. O capítulo 6 apresenta as conclusões do trabalho.

[3]<https://jekyllrb.com>

2 CONCEITOS FUNDAMENTAIS: PÁGINAS WEB

Páginas podem ser classificadas como estáticas ou dinâmicas de acordo com o seu comportamento. Neste capítulo introduzimos as noções de páginas estáticas e páginas dinâmicas e apresentamos as definições que serão usadas no restante do trabalho. Além disso, será discutido sobre vantagens e desvantagens de cada.

Páginas dinâmicas

Páginas dinâmicas são aquelas que são geradas a cada requisição e/ou contém um *script* que se comunica com o lado do servidor do site. Para uma mesma Uniform Resourse Locator (URL), arquivos diferentes podem ser entregues a clientes diferentes e um mesmo cliente pode receber arquivos diferentes em momentos diferentes (TANENBAUM; WETHERALL, 2011). Quando um cliente faz uma requisição para uma página dinâmica, o servidor passa os parâmetros da requisição a um *script* que, baseado nos parâmetros, gera uma página HTML. Um exemplo é quando o *script* consulta um ou mais bancos de dados para obter o conteúdo da página e passa o resultado da consulta para uma *engine* de *template*, que irá montar uma página HTML utilizando o conteúdo e os *templates* (TANENBAUM; WETHERALL, 2011).

A figura 2.1 mostra uma página dinâmica gerada no servidor. Após a requisição do cliente (1), o lado do servidor do site (identificado como **Site**) gera a página (2), que não existe previamente, e depois a entrega ao cliente (3). O passo (2) (geração da página) configura comportamento dinâmico.

A figura 2.2 mostra uma página dinâmica que se comunica com o lado do servidor do site. Nos passos (3) e (4), a página se comunica com o lado do servidor do site (identificado como **Site**) o que configura comportamento dinâmico. Isso independe de a página ter sido gerada a partir de uma requisição ou já existir previamente no servidor.

Páginas estáticas

Uma página estática é basicamente uma coleção de arquivos (e.g. páginas no formato HTML, imagens no formato JPG, arquivos de estilo no formato CSS) que, a cada requisição, é entregue ao cliente. Os arquivos são entregues assim como existem no ser-

Figura 2.1: Página dinâmica com geração no servidor

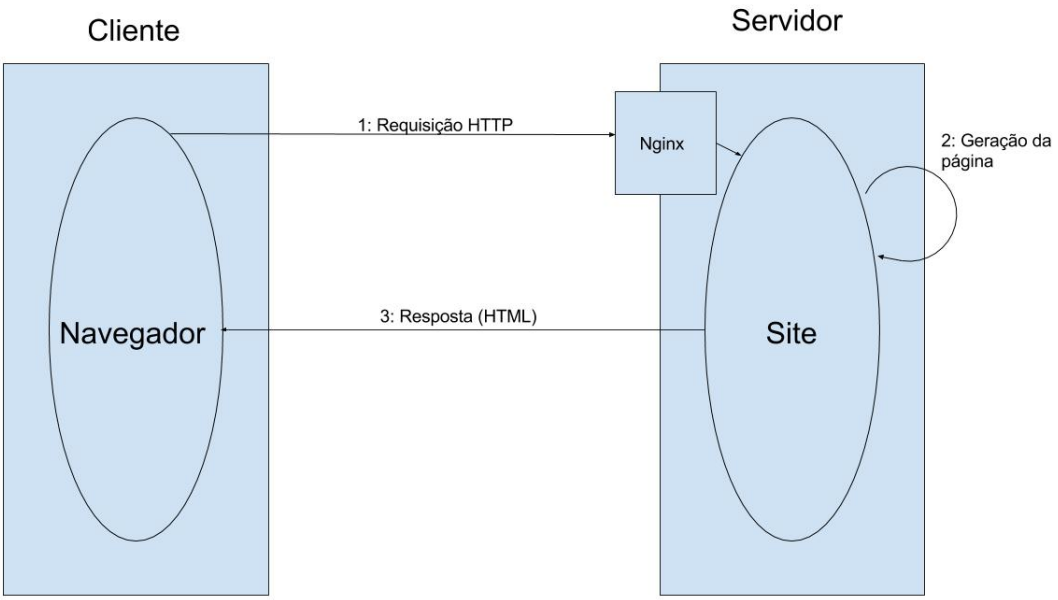


Figura 2.2: Página dinâmica com comunicação com o lado do servidor do site

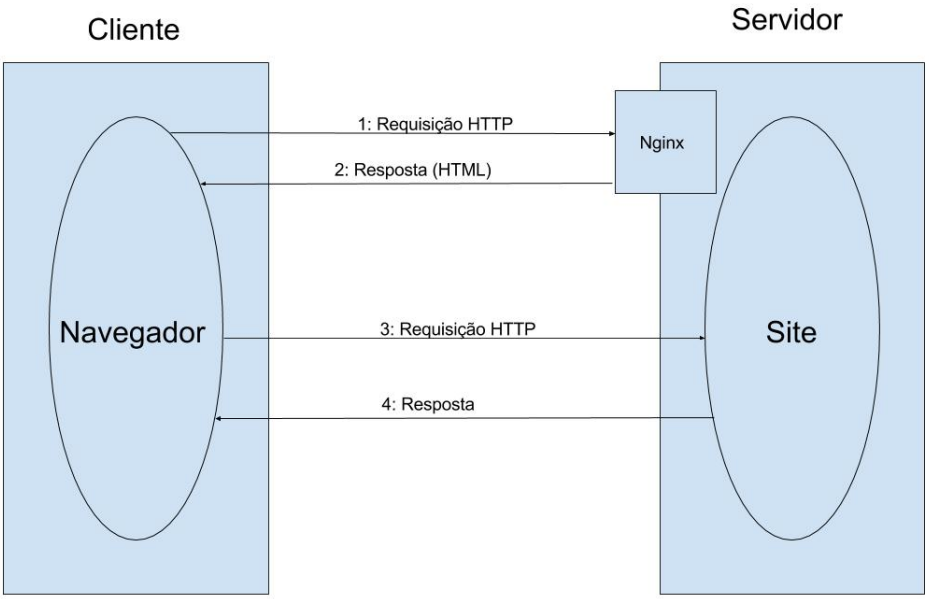
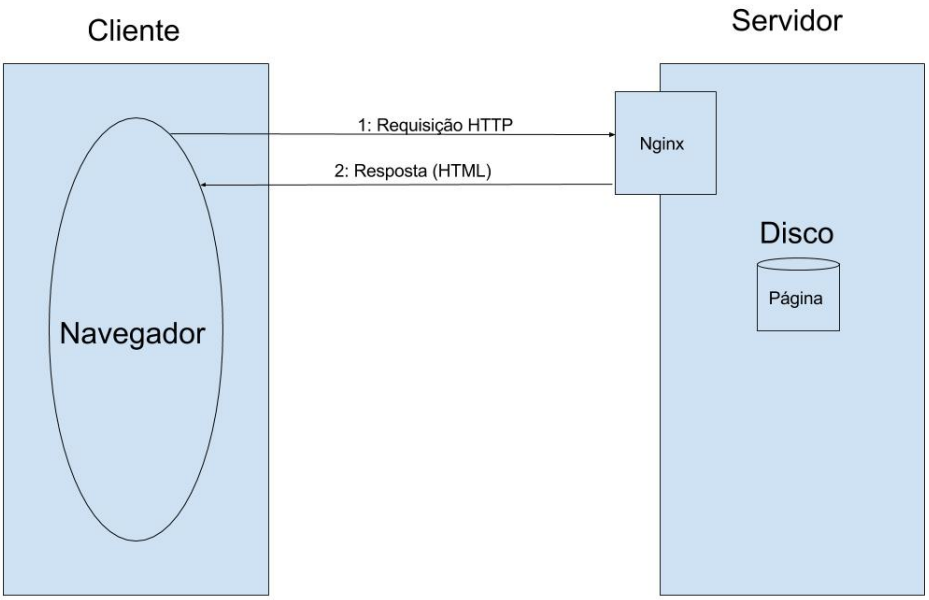


Figura 2.3: Página estática

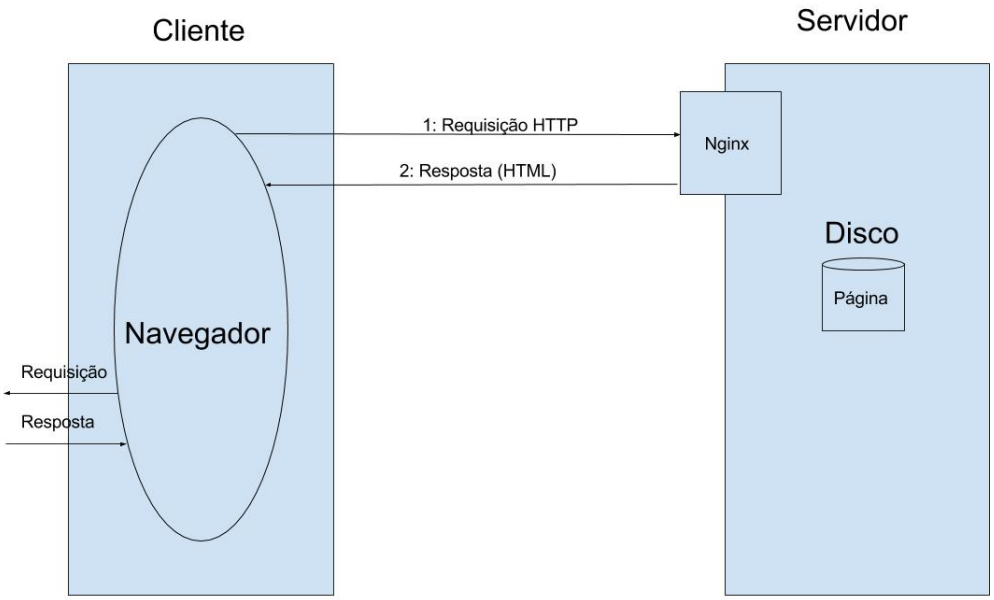


vidor, sem processamento. Para uma mesma URL, clientes diferentes recebem a mesma página, e o mesmo cliente recebe a mesma página em momentos diferentes (TANENBAUM; WETHERALL, 2011)., a não ser que o conteúdo dos arquivos seja alterado no servidor entre requisições. O simples fato de *scripts* executarem no cliente não torna a página dinâmica. Uma página pode ter um *script* rodando no cliente, mas continua sendo estática desde que não seja gerada a partir de uma requisição, nem no cliente e nem no servidor, e desde que não se comunique com o lado do servidor do seu site (caso exista) (RINALDI, 2015).

A figura 2.3 mostra uma página estática simples. A página já existe no disco do servidor, pronta para ser entregue. Ao receber uma requisição (1), o servidor simplesmente entrega a página ao cliente (2).

A figura 2.4 mostra uma página estática que se comunica com servidores de terceiros. A página já existe no disco do servidor, pronta para ser entregue, assim como na página estática da figura 2.3. A diferença nesse exemplo é que a página realiza comunicação com outros servidores. Por definição, como a página não é gerada a partir de uma requisição e não se comunica com o lado do servidor do seu site, é estática.

Figura 2.4: Página estática com comunicação com servidores de terceiros



Sites

Um site estático é aquele que tem apenas páginas estáticas e não contém um *script* ou programa rodando no servidor. Um site dinâmico é aquele que possui ao menos uma página dinâmica e/ou possui um *script* ou programa rodando no servidor.

Vantagens e desvantagens de páginas dinâmicas

Grande parte do uso da Web envolve aplicações e serviços, e.g., *e-commerce*, buscas em catálogos de livros ou músicas, navegação em mapas e colaboração em documentos. Programas executando no servidor e comunicação do cliente com o servidor são características de páginas dinâmicas, e permitem que páginas Web sejam utilizadas para aplicações e serviços. Páginas estáticas não permitem, por exemplo, a implementação de uma aplicação de *e-commerce* (TANENBAUM; WETHERALL, 2011).

Páginas dinâmicas são muito atraentes por poderem personalizar conteúdo, oferecer sistema de login, seção de comentários e interação com o usuário. *Features* como essas, que fazem com que potencialmente para uma mesma URL sejam retornados arquivos HTML diferentes, não podem ser implementadas em um site puramente estático.

No entanto, sites dinâmicos também possuem desvantagens. O processamento en-

volvido é elevado, uma vez que o servidor precisa gerar uma página para cada requisição. Em picos ou crescimento de tráfego, esse processamento pode ser um problema. Uma forma de contornar isso é utilizando cache, disponibilizando arquivos HTML que podem ser requisitados novamente no servidor, mas a própria gerência de cache para páginas web pode ser uma tarefa complexa, e não há garantia de que duas requisições à mesma URL retornarão o mesmo HTML (RINALDI, 2015).

Vantagens de sites estáticos

Velocidade

Por não haver acesso a banco de dados, geração de arquivos com *engines* ou processamento para cada requisição, os sites se tornam mais rápidos. O site inteiro consiste em arquivos estáticos, que são servidos a cada requisição. Isso também significa que, em picos de tráfego, sites estáticos têm um desempenho melhor que sites dinâmicos (RINALDI, 2015).

Segurança

Um site que possui sistema de login, processamento para cada requisição ou acesso a banco de dados oferece vulnerabilidades em potencial. Sites estáticos eliminam muitas vulnerabilidades com a sua simplicidade, uma vez que restringem a comunicação entre servidor e cliente ao envio de arquivos estáticos do servidor para o cliente (RINALDI, 2015).

Versionamento de Conteúdo

Ferramentas de versionamento de código, como git, são muito populares e, muitas vezes, até essenciais para projetos complexos. Com sites estáticos, o próprio conteúdo pode ser versionado, uma vez que pode estar em arquivos que fazem parte do código ao invés de estar em bancos de dados. Isso permite que, por exemplo, outras pessoas possam propor correções ou melhorias ao conteúdo através de *pull requests* (RINALDI, 2015).

Simplicidade do Servidor

Um site estático não precisa ter um programa ou *script* no servidor fazendo algum tipo de processamento, uma vez que ele é uma coleção de arquivos estáticos. Isso simplifica a infraestrutura. Com geradores estáticos, após os arquivos serem gerados e estarem no servidor, nada mais precisa ser feito além de servi-los a cada requisição (RINALDI, 2015).

Desvantagens de sites estáticos

Sem conteúdo em tempo real

A não ser que o site seja gerado novamente, uma mesma URL sempre retornará o mesmo arquivo HTML. Não há conteúdo personalizado, não há pesquisa em grandes coleções de itens em um banco de dados, não há interação do usuário com o servidor. É impossível, na prática, implementar uma loja online que lista produtos em um resultado de busca de forma estática, a não ser que a lista total de itens seja pequena o suficiente para ser enviada por completo ao cliente dentro da página.

Sem entrada de usuário

É possível que o usuário interaja com uma página estática que possui um script, mas não há um programa no servidor que processa entrada de usuário, isso significa que não é possível, por exemplo, ter um sistema de comentários ou sistema de login de forma estática. Colaboração em documentos também não é uma possibilidade, navegação em mapas tampouco.

As desvantagens limitam consideravelmente o uso de sites estáticos, uma vez que, caso seja necessária ao menos uma funcionalidade dinâmica no site, ele não pode ser implementado de forma estática. Existem soluções alternativas que permitem que um site estático tenha funcionalidades dinâmicas, no entanto isso diminui a sua simplicidade, uma das suas vantagens, e podem ser soluções apenas parciais. Em muitos casos, o cenário é proibitivo para um site estático ou simplesmente é mais conveniente utilizar um site dinãmico.

Soluções alternativas

Há algumas formas de incluir algumas funcionalidades em páginas estáticas que, a princípio, só seriam possíveis em páginas dinâmicas. Isso é possível com a combinação da possibilidade de *scripts* no cliente com serviços recentes oferecidos por terceiros. Para seção de comentários, pode-se utilizar Disqus[1] ou Facebook[2]. Com *plugins* do Twitter[3] ou Facebook[4], se pode integrar a rede social no site. Com o Firebase[5], é possível ter um banco de dados em tempo real como serviço, que fica armazenado na nuvem do Firebase. Swiftype[6] provê serviço de busca. Olark[7] permite adicionar um chat de suporte em tempo real. Formulários podem criados com o serviço do Google[8].

O surgimento desses serviços contribui para o crescimento de popularidade de sites estáticos[9]. Eles estendem a capacidade de sites estáticos de forma relativamente simples, basta o desenvolvedor incluir um *script* do fornecedor do serviço na página desejada e não precisa se preocupar com todos os detalhes de implementação da funcionalidade em questão. Isso quer dizer que, em muitos casos onde o desenvolvedor optava por uma alternativa dinâmica, por desejar ou precisar de funcionalidades como essas descritas aqui, agora o desenvolvedor pode utilizar um site estático.

[1] <https://disqus.com>
[2] <https://developers.facebook.com/docs/plugins/comments>
[3] <https://dev.twitter.com/web/overview>
[4] <https://developers.facebook.com/docs/plugins>
[5] <https://firebase.google.com>
[6] <https://swiftype.com>
[7] <https://www.olark.com>
[8] <https://www.google.com/forms/about>
[9] <https://www.smashingmagazine.com/2015/11/modern-static-website-generators-next-big-thing>

3 SISTEMAS DE GERENCIAMENTO DE CONTEÚDO

Este capítulo apresenta os sistemas de gerenciamento de conteúdo, que são ferramentas muito populares para a criação e manutenção de sites dinâmicos atualmente. São citados os três CMS mais populares, WordPress, Joomla! e Drupal. Também são discutidos alguns de seus pontos positivos e negativos.

CMS

WordPress (WP) é CMS baseado em PHP e MySQL[1]. Em dezembro de 2017, WP era usado por mais de 29% dos 10 milhões de sites mais populares[2], e era o CMS mais utilizado na Web[3].

O WP oferece um painel de controle de onde o usuário pode gerenciar o seu site. Isso se dá através de uma interface gráfica, como mostrado na figura 3.1. A partir do painel de controle, o usuário pode fazer diversas tarefas como criar novas publicações, editar as páginas do site, gerenciar os *plugins* e verificar estatísticas de uso do site. A facilidade do uso, juntamente com a interface gráfica e a existência de mais de 50 mil plugins[4]) para as mais diversas finalidades contribuem para a sua popularidade.

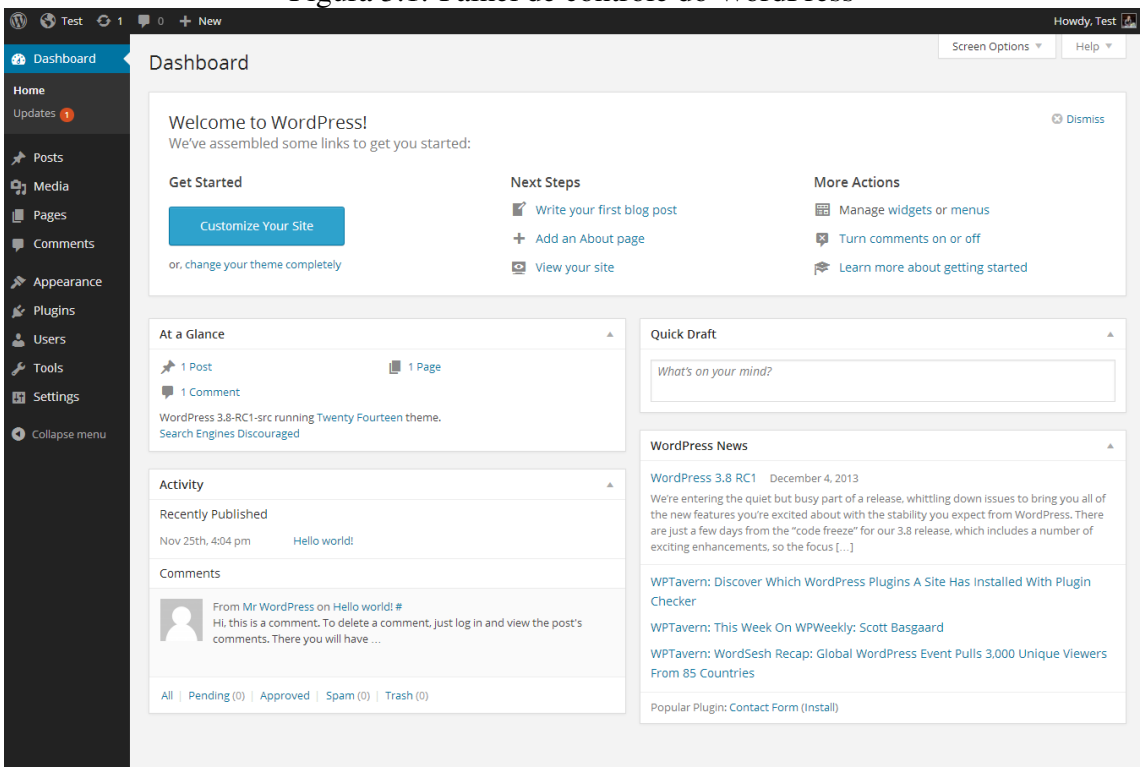
Joomla e Drupal são outros exemplos dentre os CMS mais populares. Existe um grande número de plataformas desse tipo, e grande parte da Web é composta por sites que as usam[5]. Em dezembro de 2017, Drupal possuía mais de 39 mil módulos que estendem a sua funcionalidade[6], e mais de 2 mil temas[7], enquanto Joomla possuía mais de 8 mil extensões[8].

Vantagens e desvantagens de CMS

A existência de módulos, extensões e *plugins*, e a possibilidade de se criar novos, tornam os CMS muito versáteis e convenientes, principalmente para usuários sem

[1] <https://wordpress.org/about/requirements>
[2] <https://w3techs.com/technologies/overview/content_management/all>
[3] <https://trends.builtwith.com/cms>
[4] <https://wordpress.org/plugins>
[5] <https://w3techs.com/technologies/overview/content_management/all>
[6] <https://www.drupal.org/project/project_module>
[7] <https://www.drupal.org/project/project_theme>
[8] <https://extensions.joomla.org>

Figura 3.1: Painel de controle do WordPress



Fonte: WordPress (<https://codex.wordpress.org/Dashboard_Screen>)

habilidades de desenvolvimento de software. Esses adicionais, que são instalados separadamente do núcleo do CMS, fazem desde consulta a banco de dados até mudar a aparência do site. Essa arquitetura permite que alguém sem conhecimento de programação possa criar sites diversos. Inclusive, pessoas que talvez desistissem de criar um site por falta de conhecimento ou por receio de investir em um site tradicional (sem CMS), optam por criar o seu site em um CMS.

No entanto, essa flexibilidade toda tem as suas desvantagens. Cada um dos *plugins* de uma instalação vai ser carregado juntamente com o núcleo do CMS, podendo ocasionar queda de desempenho por causa da quantidade grande de partes executando no servidor. Além disso, como os *plugins* podem realizar processamento arbitrário, é possível que em uma requisição por uma página, por exemplo, diversos *plugins* realizem alguma tarefa relacionada com a sua aparência e conteúdo, podendo inclusive fazer acesso ao banco de dados. Isso faz com que o tempo de carregamento da página seja afetado, e é uma consequência direta do fato de as páginas serem geradas para cada requisição. As páginas também podem se comunicar com o lado do servidor do site durante a sua interação com o usuário, podendo afetar ainda mais o desempenho e tornar a página menos responsiva.

Usuários da Web são impacientes, e têm expectativas altas com relação ao tempo

de resposta de um site. Uma demora de mais do que alguns segundos já pode levar à frustração. De acordo com (NAH, 2004), usuários da Web esperam um tempo de resposta em torno de 2 segundos para uma simples obtenção de informação, e esse mesmo tempo de resposta é necessário para garantir uma interação suave entre a Web e os usuários. Além disso, um tempo de carregamento ou um *delay* muito alto podem fazer com que o usuário perca a sua atitude cooperativa com o site. A demora também pode fazer com que a pessoa abandone a tarefa em questão ou que decida não retornar ao site novamente. (GALLETTA et al., 2004) observou que o *delay* de um site deve permanecer abaixo de 8 segundos para incentivar uma atitude positiva do usuário, e se o objetivo é fazer com que o usuário persista em uma dada tarefa ou retorne posteriormente, esse *delay* deve ficar abaixo de 4 segundos. Isso pode simplesmente eliminar a possibilidade de se utilizar um CMS para fazer um determinado site.

É impossível, atualmente, saber de todas as vulnerabilidades existentes nos *plugins* do WP e outros CMS. A facilidade de se criar novos *plugins* ou extensões e torná-los disponíveis faz com que muitas pessoas, mesmo sem as devidas precauções de segurança, desenvolvam esses módulos de forma pouco responsável. E como as instalações de CMS costumam utilizar diversos *plugins* e extensões, a tarefa de verificar cada um por vulnerabilidades se torna mais difícil. (FONSECA; VIEIRA, 2014) encontrou mais de 350 vulnerabilidades previamente desconhecidas em 35 *plugins* do WP, das quais 138 podem ser consideradas fáceis de explorar por serem diretamente relacionadas com *input* de usuário.

Apesar dos vários times de segurança existentes, não necessariamente todas as brechas são resolvidas a tempo de garantir a segurança das instalações. Segundo (JERKOVIC; SINKOVIC, 2017), as principais entidades responsáveis por lidar com segurança dos CMS não possuem uma estratégia de resposta devidamente organizada em caso de vulnerabilidades recentemente descobertas.

Infelizmente, confiar na opinião da comunidade não é uma abordagem livre de falhas. Nem sempre os usuários têm uma noção correspondente à realidade sobre a segurança dos *plugins* que utilizam. (KOSKINEN; IHANTOLA; KARAVIRTA, 2012) encontraram uma correlação fraca entre estimativas de usuários e vulnerabilidades verificadas no trabalho para 332 *plugins* do WP aleatoriamente selecionados.

Mesmo uma instalação nova de WP pode oferecer brechas, de tal forma que não se pode confiar cegamente no CMS. O usuário deve ser precavido com os seus dados e não deixar a responsabilidade toda para o sistema. (KYAW; SIOQUIM; JOSEPH, 2015)

mostraram a vulnerabilidade de uma instalação nova de WP *crackeando* uma senha com um ataque de dicionário.

Isso torna claro que, ao mesmo tempo que é muito atrativo criar um site usando um CMS com diversos *plugins* e extensões sem a necessidade de saber programar, essa possibilidade é perigosa, uma vez que o criador do site pode cometer o erro de incluir muitos *plugins*, possivelmente alguns dos quais não confiáveis, e tornar o seu site lento e vulnerável, ou criar uma aplicação que simplesmente não funciona. Também se observa que, mesmo em um cenário de aplicações Web amplamente testadas e usadas por um grande número de pessoas, como WP, *plugins* são uma fonte de vulnerabilidades em potencial.

O estado atual das coisas aponta para a importância dos usuários verificarem as extensões que utilizam, preferencialmente rodando testes, e não apenas confiando na opinião da comunidade, que não necessariamente corresponde à realidade. Isso causa um certo impasse, já que grande parte dos usuários de CMS não são programadores, e justamente se beneficiam do fato de não precisarem saber o que se passa por baixo dos *plugins*, mas apenas o serviço que eles oferecem.

4 GERADORES DE SITES ESTÁTICOS

Este capítulo apresenta os geradores de sites estáticos, ferramentas que crescem em popularidade atualmente, e são usadas para criação e gerência de sites estáticos. São mostrados alguns detalhes do Jekyll, o gerador mais popular e com a comunidade mais ativa.

Os geradores

Geradores de websites estáticos, ou geradores de sites estáticos (GSE), são programas que, a partir de *template*, configuração, conteúdo e dados, geram um site composto por objetos estáticos, como HTML, CSS e JPEG.

Os *templates* consistem em arquivos escritos em linguagens de *template*, como Liquid. A configuração utiliza alguma linguagem de serialização, como YAML. Os dados podem ser representados de diversas formas, como em JSON, CSV ou YAML. O conteúdo é representado em alguma linguagem de *markup*, como HTML ou Markdown, e cada arquivo de conteúdo gera uma página no site. A tabela a seguir mostra alguns dos geradores mais populares e a linguagem na qual foram construídos.

Geradores de sites estáticos crescem em popularidade atualmente. O site StaticGen[1] lista GSE que têm seu repositório no GitHub, e mostra o crescimento no número de estrelas e *forks* dos projetos. Mas apesar de estarem crescendo em popularidade, os GSE

[1] <https://www.staticgen.com>

Tabela 4.1: Alguns dos geradores de sites estáticos mais populares ordenados por número de estrelas do seu repositório no GitHub e a linguagem na qual foram construídos

Gerador	Linguagem
Jekyll	Ruby
Hugo	Go
Hexo	JavaScript
GitBook	JavaScript
Gatsby	JavaScript
Nuxt	JavaScript
Pelican	Python
Metalsmith	JavaScript
Brunch	JavaScript

Fonte: StaticGen (<https://www.staticgen.com>)

exitem há mais tempo. O site Static Site Generators[2], que também lista GSE, contendo mais de 450 geradores listados, mostra que alguns existem há 15 anos.

De qualquer forma, os geradores mais recentes costumam ter algumas características em comum: rodar pela linha de comando, suportar pelo menos uma linguagem de *template*, prover um servidor local para testes, suportar formatos de dados em arquivos, ter uma arquitetura extensível e ter um processo de *build*(RINALDI, 2015). A seguir esses pontos são observados em mais detalhe.

Rodar pela linha de comando

Em geral, geradores de sites estáticos não oferecem interface gráfica. Praticamente todo o trabalho é feito pela linha de comando, a geração dos arquivos padrão no início de um novo projeto, execução de um servidor local de testes, o *build* e possivelmente também o *deploy*.

Para iniciar um novo projeto com o Jekyll, o mais popular dos geradores, os seguintes comandos devem ser executados no terminal:

```
gem install jekyll bundler
jekyll new meu-novo-site
```

(O comando *gem* requer que o *RubyGems* esteja instalado na máquina. Caso não esteja, pode ser baixado no site do RubyGems[3])

'gem install jekyll bundler' instala as *gems* **jekyll** e **bundler** através do *RubyGems*. **bundler** é uma *gem* que gerencia outras *gems*. Ela garante que as *gems* e as versões sejam compatíveis, e que todas as dependências necessárias para cada *gem* estejam instaladas. **jekyll** é a *gem* que provê todo o necessário para a criação de projetos com Jekyll. As duas precisam ser instaladas apenas uma vez, e depois estarão disponíveis para uso a qualquer momento no sistema operacional onde foram instaladas.

Esses comandos criam um novo site e todos os arquivos básicos necessários para começar o projeto. A partir disso, já se pode mudar para o diretório recém criado e executar o servidor local de testes:

```
cd meu-novo-site
bundle exec jekyll serve
```

[2]<https://staticsitegenerators.net>
[3]<https://rubygems.org/pages/download>

Neste momento, o servidor está rodando e pode-se acessar o site em ‘http://localhost:4000‘.

O comando ‘bundle exec jekyll serve‘ gera o site e roda um servidor para a sua visualização em localhost. Por padrão, ele verifica quando ocorrem mudanças nos arquivos de origem e gera o site novamente quando elas existem. O comando ‘bundle exec jekyll build‘ também pode ser usado para gerar o site, nesse caso sem rodar um servidor local.

Esses comandos podem ser executados sem a parte ‘bundle exec‘, i.e., ‘jekyll serve‘ ou ‘jekyll build‘. No entanto, o **bundler**, ao garantir que as versões das *gems* e suas dependências estejam corretas, previne que os comandos falhem em função disso, de tal forma que é mais simples utilizá-lo.

Linguagem de *template* para a temática

Os GSE permitem a criação de temáticas para o site facilmente através de linguagens de *template*. Isso permite que o desenvolvedor defina a aparência do site e como e onde o conteúdo vai ser apresentado. Normalmente, os GSE suportam linguagens já existentes. Por exemplo, Jekyll usa por padrão a linguagem de *template* Liquid, enquanto Hexo usa a linguagem Swig. Essas e outras linguagens, como EJS e Jade, são usadas por padrão ou suportadas por diversos GSE. O repositório ’Static-Site-Samples’, do ’remotesynth’ no GitHub[4] (visitado em 29 11 2017) contém exemplos de diversos GSE, utilizando-se de linguagens de *template* diferentes, implementando o mesmo site.

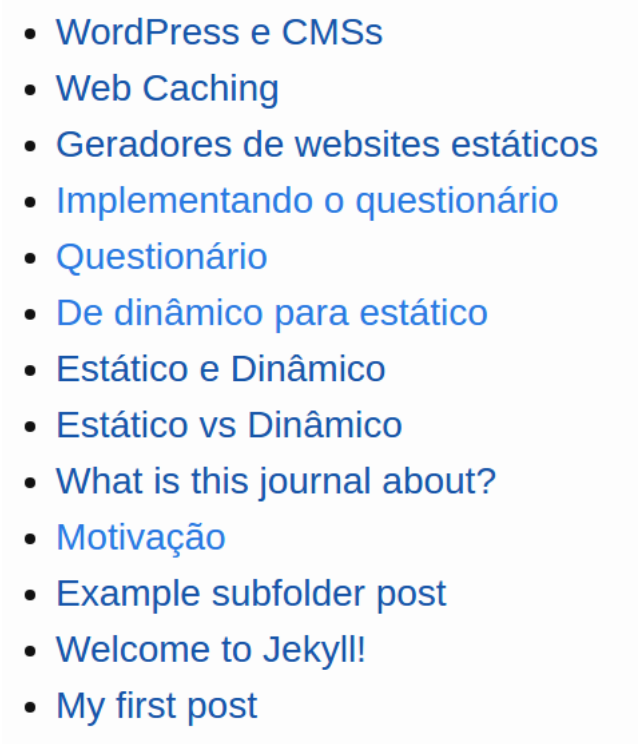
A seguir está um exemplo, com a linguagem Liquid de como criar uma lista do título de todos os *posts* de um blog feito com Jekyll, onde cada item é um link para o respectivo post. O código está dentro de uma página do blog, e a lista de títulos é gerada naquela página.

```
<ul>
  {% for post in site.posts %}
    <li>
      <a href="{{ post.URL }}">{{ post.title }}</a>
    </li>
  {% endfor %}
</ul>
```

O resultado do código é mostrado na figura 4.1.

[4] <https://github.com/remotesynth/Static-Site-Samples>

Figura 4.1: Página inicial do site acessada pelo servidor local



Servidor local para testes

A maioria dos GSE incluem um servidor local que pode ser rodado pela linha de comando a partir do diretório do site. Isso é muito útil para o desenvolvimento, já que o site não precisa ser gerado cada vez que o desenvolvedor quer visualizar mudanças. Inclusive, a maioria dos servidores locais possui uma funcionalidade de geração automática, ou seja, eles ficam executando e, ao perceber mudanças nos arquivos do site, automaticamente geram o site novamente. Isso significa que o desenvolvedor não precisa nem gerar o site manualmente para a visualização local, e que a cada mudança já pode observar os resultados no navegador.

Abaixo está um exemplo de como rodar o servidor local do Jekyll (utilizando o **bundler**) pela linha de comando. Primeiro, o modo padrão:

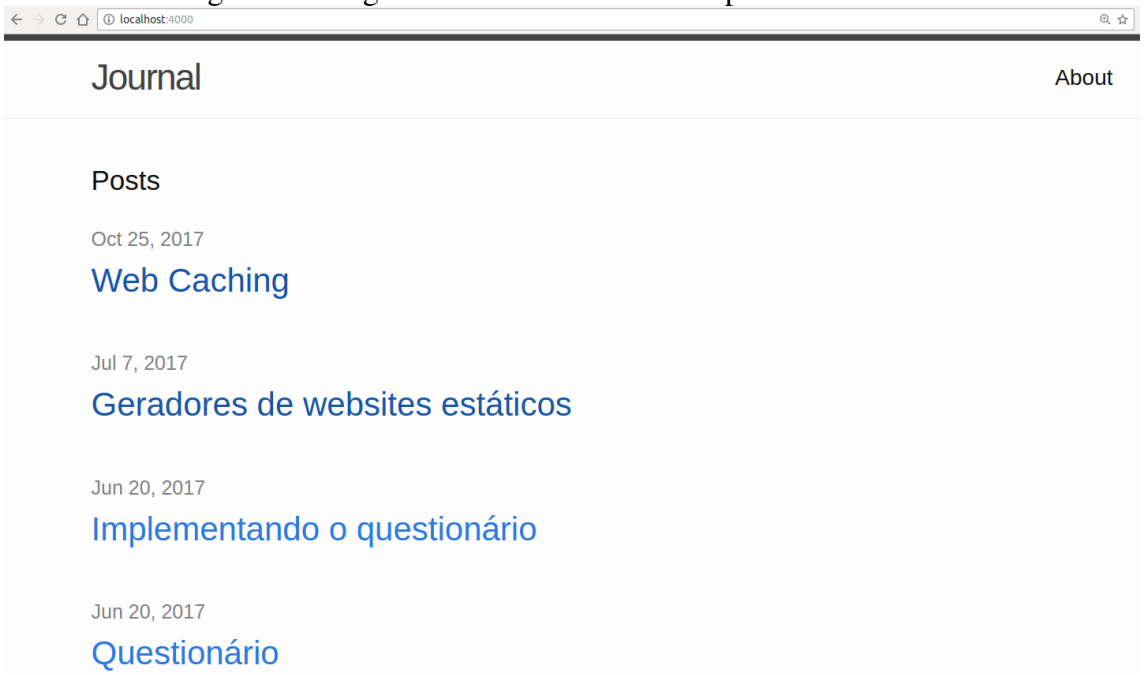
```
bundle exec jekyll serve
```

E aqui um exemplo do modo sem a geração automática.

```
bundle exec jekyll serve --no-watch
```

A figura 4.2 mostra o site sendo acessado no navegador.

Figura 4.2: Página inicial do site acessada pelo servidor local



Formatos de dados em arquivos

Em geral, GSE suportam ao menos um formato de dados em arquivos, como YAML, JSON ou CSV. Esses formatos permitem que os dados estejam em arquivos ao invés de em bancos de dados, e isso é interessante porque elimina a necessidade de um banco de dados para o site e permite que os dados façam parte do código, podendo estar no repositório e serem modificados da mesma forma que o restante do código é modificado.

Isso permite que qualquer tipo arbitrário de dados possa ser representado de forma independente da forma como vai ser apresentado no site. Por exemplo, com Jekyll, se tivermos um arquivo chamado ‘questions.json‘ dentro da pasta ‘_data‘, que contém um array de objetos, onde cada objeto contém uma chave chamada ‘question‘, podemos listar as questões do questionário da seguinte forma[5]:

```
<ul>
{% for item in site.data.questions %}
  <li>
    {{ item.question }}
  </li>
{% endfor %}
```

[5]<https://jekyllrb.com/docs/datafiles>

Figura 4.3: Parte da lista de questões gerada pelo código Liquid

- {"id"=>1, "name"=>"Task", "cat"=>1, "question"=>"Mobility of the task", "values"=>[{"name"=>"Low", "num"=>0, "cri"=>1}, {"name"=>"Medium", "num"=>1, "cri"=>1}, {"name"=>"High", "num"=>2, "cri"=>1}]}
- {"id"=>2, "name"=>"Actor", "cat"=>1, "question"=>"Mobility of the Actor", "values"=> [{"name"=>"Low", "num"=>0, "cri"=>2}, {"name"=>"Medium", "num"=>1, "cri"=>2}, {"name"=>"High", "num"=>2, "cri"=>2}]}
- {"id"=>3, "name"=>"Frequency", "cat"=>2, "question"=>"Number of executions", "values"=> [{"name"=>"Rarely", "num"=>0, "cri"=>3}, {"name"=>"Regularly", "num"=>1, "cri"=>3}, {"name"=>"Often", "num"=>2, "cri"=>3}]}
- {"id"=>4, "name"=>"Urgency", "cat"=>2, "question"=>"Importance of performing the task immediately", "values"=>[{"name"=>"Low", "num"=>0, "cri"=>4}, {"name"=>"Medium", "num"=>1, "cri"=>4}, {"name"=>"High", "num"=>2, "cri"=>4}]}
- {"id"=>5, "name"=>"Digitalization", "cat"=>2, "question"=>"Potential of digitalization", "values"=> [{"name"=>"Low", "num"=>0, "cri"=>5}, {"name"=>"Medium", "num"=>1, "cri"=>5}, {"name"=>"High", "num"=>2, "cri"=>5}]}
- {"id"=>6, "name"=>"Devices", "cat"=>2, "question"=>"Possibilities to replace other devices with mobile touch-based devices", "values"=>[{"name"=>"Low", "num"=>0, "cri"=>6}, {"name"=>"Medium", "num"=>1, "cri"=>6}, {"name"=>"High", "num"=>2, "cri"=>6}]}
- {"id"=>7, "name"=>"Usability", "cat"=>2, "question"=>"Improvements of usability through mobile touch-based devices", "values"=>[{"name"=>"Low", "num"=>0, "cri"=>7}, {"name"=>"Medium", "num"=>1, "cri"=>7}, {"name"=>"High", "num"=>2, "cri"=>7}]}
- {"id"=>8, "name"=>"Sensors", "cat"=>2, "question"=>"Enrichment of the application through the use of sensors", "values"=>[{"name"=>"No", "num"=>0, "cri"=>8}, {"name"=>"Yes", "num"=>2, "cri"=>8}]}
- {"id"=>9, "name"=>"Data Volume Transmit", "cat"=>3, "question"=>"Amount of data which have to be transmitted", "values"=>[{"name"=>"GB", "num"=>0, "cri"=>9}, {"name"=>"MB", "num"=>1, "cri"=>9}, {"name"=>"KB", "num"=>2, "cri"=>9}]}
- {"id"=>10, "name"=>"Data Volume Receive", "cat"=>3, "question"=>"Amount of data which have to be received", "values"=>[{"name"=>"GB", "num"=>0, "cri"=>10}, {"name"=>"MB", "num"=>1, "cri"=>10}, {"name"=>"KB", "num"=>2, "cri"=>10}]}
- {"id"=>11, "name"=>"Computing Power", "cat"=>3, "question"=>"Amount of computing power the application requires", "values"=>[{"name"=>"High", "num"=>0, "cri"=>11}, {"name"=>"Medium", "num"=>1, "cri"=>11}, {"name"=>"Low", "num"=>2, "cri"=>11}]}

Como resultado, a lista de questões é mostrada na página. Parte da lista aparece na imagem 4.3.

Arquitetura extensível

Diversos GSE oferecem plug-ins ou extensões para adicionar funcionalidade ou customizar o comportamento do gerador. Em geral, quanto maior a comunidade de usuários, maior a lista de plug-ins disponíveis. É possível criar novas extensões ou plug-ins para esses GSE, mas isso exige conhecimento da linguagem na qual o gerador foi construído. Jekyll oferece uma página na documentação sobre como contribuir com o projeto,

incluindo a adição de novos plug-ins[6]

Processo de build

Essa é a principal característica de um GSE: com um comando, gerar um site estático. Para gerar um site com o Jekyll, o seguinte comando é utilizado:

```
jekyll build
```

O comando deve ser executado no diretório onde o site foi criado com ‘jekyll new‘. E para executá-lo com o **bundler** temos o seguinte:

```
bundle exec jekyll build
```

Como já citado anteriormente, o comando ‘jekyll serve‘ também realiza o processo de build. Esses comandos podem ser customizados pela linha de comando e por arquivos de configuração.

[6]<https://jekyllrb.com/docs/contributing>

5 A IMPLEMENTAÇÃO DE UMA PÁGINA DINÂMICA COM O JEKYLL

Neste capítulo consideramos uma página Web dinâmica (que chamaremos de página original) e analisamos o seu funcionamento e quais eventos ou funcionalidades a tornam uma página dinâmica. Em seguida, analisamos quais dessas funcionalidades podem ser implementadas de forma estática. Então, descrevemos como a página foi implementada utilizando o Jekyll, um GSE, com essas funcionalidades implementadas de forma estática. Por fim, detalhamos o comportamento da página resultante.

A página original

A página em questão faz parte de um site que foi implementado utilizando NodeJS no servidor e JavaScript com AngularJS no cliente. Ela implementa um questionário, onde é apresentada ao usuário uma sequência de questões de múltipla escolha para que ele selecione uma das possíveis respostas e clique em um botão para confirmar e seguir para a próxima questão. A figura 5.1 mostra o questionário em execução.

As questões ficam armazenadas em um banco de dados SQL (SQLite), e são enviadas do servidor ao cliente como resposta para uma requisição HTTP. Um *script* no cliente é responsável por fazer a requisição, executar o questionário e processar as respostas, que são, através de mais uma requisição, enviadas ao servidor e armazenadas no banco de dados.

A sequência de eventos é mostrada na figura 5.2: o cliente faz uma requisição HTTP para o servidor, pedindo pela página (1). Isso dispara, no servidor, a geração da página, que é processada por uma *engine* de *template* (2). Quando gerada, a página é entregue ao cliente (3), que mostra a página ao usuário no navegador. O *script* contido na página, então, faz uma nova requisição ao lado do servidor do site (representado por **Site**), pedindo pelas questões (4). O servidor, ao receber a requisição, consulta o banco de dados, obtém as questões e as entrega ao cliente (5), em forma de objeto JSON. O cliente, ao receber esse objeto, que é uma lista de questões, cada qual com as respectivas opções de resposta, as apresenta ao usuário em ordem, de uma por uma. Quando o usuário seleciona uma das respostas e clica no botão "Submit", essa resposta é processada e a próxima questão é apresentada ao usuário. Quando o usuário termina de responder todas as questões, o cliente envia um objeto JSON contendo as respostas (6).

Figura 5.1: Questionário em execução

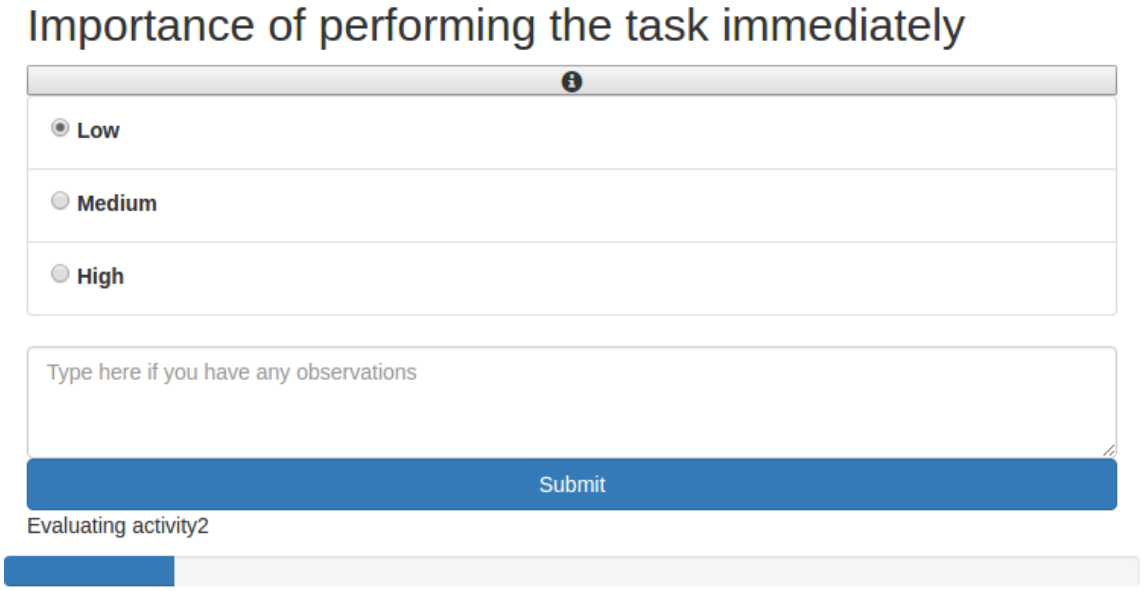
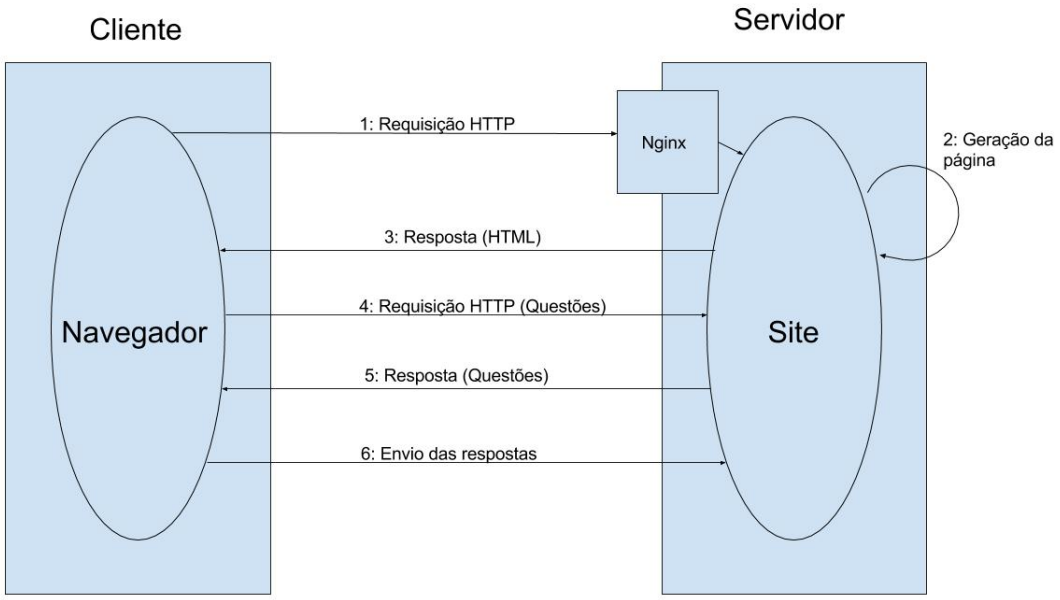


Figura 5.2: Comportamento da página original



Análise da página original

Para podermos implementar a página com o Jekyll, vamos analisar exatamente quais dos eventos descritos são dinâmicos e quais são estáticos. A requisição da página pelo cliente não caracteriza comportamento estático ou dinâmico. A geração da página no servidor, por sua vez, faz parte da definição de páginas dinâmicas, essa é uma funcionalidade dinâmica. A apresentação da página no navegador não diz nada em especial. O próximo a acontecer é uma requisição partindo da própria página, através do seu script, ao servidor, pedindo pelas questões. Isso é funcionalidade dinâmica. Também é funcionalidade dinâmica o lado do servidor receber requisições e as processar de alguma forma, como nesse caso, incluindo consulta a banco de dados, e enviar resposta ao cliente. Do momento em que o cliente tem posse das questões até o momento onde o usuário finaliza o questionário, não há comunicação com o servidor, portanto o comportamento é estático. O próximo a acontecer é mais uma conexão HTTP do cliente com o servidor, dessa vez para enviar as respostas, que são armazenadas, então, no banco de dados pelo servidor. Isso é comportamento dinâmico.

Quais das funcionalidades descritas podemos implementar de forma estática? Como precisamos de um lado do servidor do site processando as respostas e agregando respostas de diferentes usuários, não podemos nos livrar da última comunicação do cliente com o servidor, que é a que envia as respostas. No entanto, há duas coisas que podem ser implementadas de forma estática. A primeira é a geração da página no servidor para cada requisição. Todos os clientes recebem a mesma página, não há personalização. Além disso, o conteúdo da página não muda constantemente, de tal forma que ao longo do tempo os clientes continuam recebendo a mesma página ao fazer a requisição. Isso significa que não há uma necessidade fundamental de gerar a página para cada requisição. De fato, ela pode já existir no servidor como uma coleção de arquivos HTML, CSS e JavaScript e ser imediatamente entregue ao cliente quando uma requisição é feita. A segunda funcionalidade que pode ser implementada de forma estática é o envio das questões do servidor para o cliente. Ao invés de enviar as questões como resposta a uma requisição feita pela página, as questões podem estar já incluídas no próprio *script* da página como um objeto JSON, o que eliminaria completamente a necessidade de uma requisição HTTP para esse fim.

A implementação

A seguir, descreve-se como as duas funcionalidades foram implementadas de forma estática.

A geração da página a cada requisição

Qual é a alternativa a gerar a página a cada requisição? Simples, não gerar. Isso significa ter a página disponível no servidor, exatamente na forma como vai ser entregue para cada requisição. Quando um cliente pede pela página, o servidor não realiza nenhum processamento especial, não executa nenhuma *engine* de *template*, mas apenas entrega os arquivos já existentes.

Sendo assim, implementamos manualmente a página, incluindo o código JavaScript necessário, e a disponibiliza-se no servidor. No entanto, se fosse só isso, porque usar o Jekyll? Se fosse preciso implementar cada página de cada site manualmente, provavelmente se voltaria à implementação dinâmica, com as ferramentas que ajudam a criar e manter uma página web. Por isso usa-se o Jekyll, ele é uma ferramenta que facilita a criação e manutenção de um site. Um site estático. Portanto, cria-se uma página com o Jekyll, aproveitando as vantagens do GSE, e.g., aparência padrão por todo o site, organização do conteúdo e dados, versionamento de código.

Para criar uma página num projeto Jekyll, como já descrito no capítulo 4, cria-se um arquivo, em formato Markdown, dentro da pasta _posts. No início do arquivo, se descreve, em formato YAML, atributos da página, como título e data. Após isso, vem o conteúdo HTML da página, que é implementado com Markdown, HTML e Liquid (linguagem padrão de *template* do Jekyll, ver capítulo 4). O conteúdo HTML da página nada mais é do que o HTML da página original entregue ao cliente (esse HTML não existe no servidor na versão original da página, mas é gerado para cada requisição). O *script* JavaScript da página é colocado em um arquivo separado, dentro da pasta _includes, e é incluído com o seguinte código em Liquid (onde **questionnaireController.js** é o arquivo que contém o *script* da página):

```
<\textit{script} type="text/javascript">
{% include questionnaireController.js %}
</script>
```

Após feito isso, o comando **jekyll build** é executado pelo terminal no diretório do projeto Jekyll. Isso faz com que uma página HTML seja gerada a partir desses arquivos e colocada dentro da pasta _site (junto com o JavaScript) pelo Jekyll. Como consequência, temos uma página Web que pode ser colocada em um servidor via FTP e entregue assim como está para cada requisição, sem necessidade de processamento para ser gerada novamente.

Envio das questões do servidor para o cliente

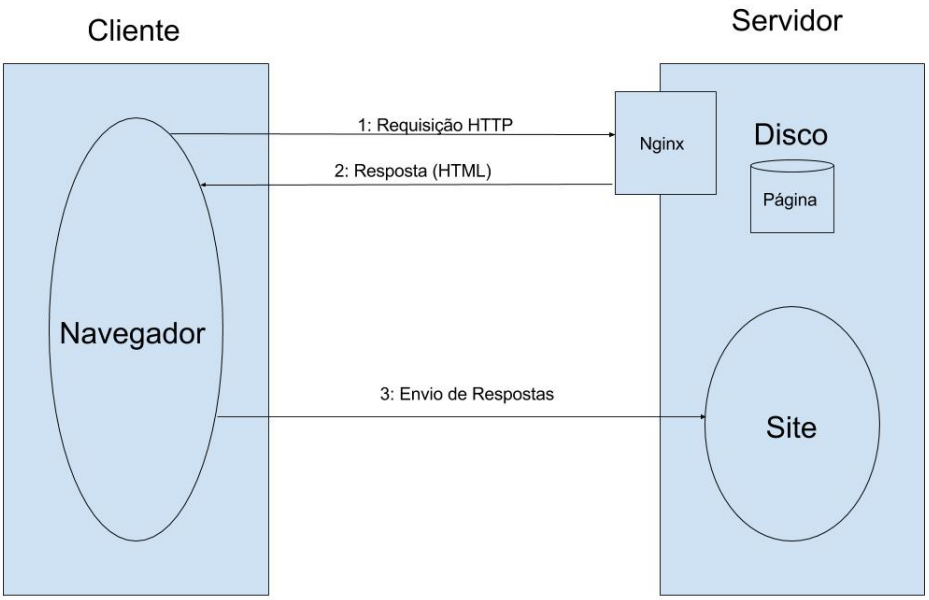
Já que as questões são as mesmas para todos os clientes, elas podem estar no próprio código da página e não precisam ser entregues a partir de uma requisição. Ou seja, no *script* da página pode-se atribuir a uma variável o objeto JSON contendo as questões assim como enviadas pelo servidor. Dessa forma, onde se usava, antes, a resposta do servidor para a requisição das questões, agora se usa essa variável.

Isso pode ser feito manualmente, atribuindo o objeto JSON das questões à variável dentro do *script* JavaScript da página. No entanto, assim como na implementação estática do primeiro item, há uma forma mais elegante de fazer isso com o Jekyll. O gerador permite, como explicado no capítulo 4, a criação de arquivos de dados, que podem ser acessados no processo de geração do site. Faz sentido usar essa funcionalidade, uma vez que dessa forma o *script* da página e o objeto JSON contendo as questões ficam em arquivos diferentes. Com isso, uma modificação às questões não gera uma modificação no arquivo que contém o *script* da página. O arquivo da página apenas inclui o arquivo de dados no seu código, através da linguagem de *template* Liquid, como mostrado no trecho a seguir:

```
<\textit{script} type="text/javascript">
var data_questions = {{ site.data.questions | jsonify }}
</script>
```

A variável **site.data.questions** é uma variável Liquid que contém os dados vindos do arquivo de dados **questions.json**. Com esse código, a variável **data_questions** pode ser acessada de dentro do *script* da página para se obter as questões. Isso permite que, por exemplo, um conjunto totalmente novo de questões possa ser usado na mesma página, mudando a referência na página para um outro arquivo de dados que contém as questões novas ou mudando o conteúdo do arquivo de dados que contém as questões. Com

Figura 5.3: Comportamento da página resultante



nenhuma ou quase nenhuma modificação ao arquivo JavaScript. Dessa maneira, o versionamento do código fica mais elegante, uma vez que as próprias questões agora fazem parte do código do site.

A implementação resultante

Implementada dessa nova forma, apesar de não ter perdido nenhuma funcionalidade, o comportamento da página mudou. Com uma geração de página no servidor a menos e uma requisição do cliente ao servidor a menos, a sequência de eventos que ocorrem desde o momento em que o cliente requisita a página até o momento em que o cliente envia as respostas ao servidor fica diferente. A seguir descrevemos o novo comportamento, que é representado na figura 5.3.

Primeiro, o cliente requisita a página ao servidor (1), que responde com arquivos HTML, CSS e JavaScript (2), sem realizar nenhum processamento extra, uma vez que a página já está disponível no disco do servidor pronta para ser entregue. Em seguida, o cliente, que já está em posse das questões (que já fazem parte do código da página), as apresenta ao usuário para que ele responda o questionário. Quando finalizado o questionário, a página abre uma nova conexão HTTP com o lado do servidor do site (representado por **Site** na figura) para entregar as respostas (3).

Tabela 5.1: Tempo para a obtenção da página original

Tempo de execução do comando **wget** (ms)
39
31
36
38
37
28
36
37
36
34
35,2

Resultados

Utilizando o comando **time** do Linux para marcar o tempo e o comando **wget** do Linux para realizar uma requisição HTTP, executamos 10 requisições para a página original e para a página reimplementada. O tempo é marcado desde o momento onde a requisição é disparada até o momento onde o cliente recebe toda a página HTML. A tabela 5.1 contém os tempos de execução para a página original, com a média de 35,2 ms no final. A tabela 5.2 contém os tempos de execução para a página que foi implementada no trabalho, com a média de 10,6 ms no final. A diferença nos tempos de resposta pode ser explicada pelo fato de que, enquanto a página original precisa ser gerada para cada requisição antes de ser entregue ao cliente, a página reimplementada já está no disco pronta para ser entregue. Os testes foram realizados em um laptop com processador Core i7-740QM e sistema operacional Ubuntu Xenial. Estratégias para performance de páginas dinâmicas como cache foram ignoradas.

Além disso, observa-se que a página resultante realiza uma requisição HTTP a menos para o lado do servidor do site. Sendo assim, há uma possibilidade a menos de ataques ao site que dependem da comunicação do cliente com o servidor. O servidor também realiza um acesso a menos ao banco de dados pelo mesmo motivo (o envio das questões ao cliente é feito no próprio código JavaScript da página), oferecendo uma vulnerabilidade em potencial a menos para ataques a banco de dados.

Tabela 5.2: Tempo para a obtenção da página resultante

Tempo de execução do comando **wget** (ms)
11
10
11
11
10
11
11
10
11
10
10,6

Aplicação

Nem todas as páginas Web podem se beneficiar da abordagem aqui apresentada. Os dois pontos a seguir descrevem duas características que indicam quando o benefício pode ocorrer para uma dada página dinâmica.

Para o primeiro ponto, suponhamos uma página que é gerada no servidor para cada requisição. Caso todas as requisições para a página em questão sejam respondidas com os mesmos arquivos, a página não precisa necessariamente ser gerada. Os arquivos (HTML, juntamente com os arquivos incluídos) podem estar previamente disponíveis no disco do servidor e serem diretamente entregues para cada requisição. Um exemplo é uma página inicial de um site dinâmico que é igual para todos os usuários e que, mesmo assim, é gerada.

Para o segundo ponto, consideremos um *script* de uma página que se comunica com o lado do servidor do seu site. Suponhamos que o *script* requisita um conjunto de dados que nunca muda. Nesse caso, esses dados já podem estar incluídos no próprio script, de tal forma que, quando a página é entregue ao cliente, os dados são juntamente entregues. Um exemplo é uma página que implementa um questionário cujas questões são sempre as mesmas e que, apesar disso, são entregues ao cliente como resposta a uma requisição HTTP feita pela página.

6 CONCLUSÃO

Muitas *features* são implementadas em páginas Web de forma dinâmica, não porque precisam necessariamente ser implementadas de forma dinâmica, mas por conveniência e/ou porque fazem parte de uma página dinâmica ou de um site dinâmico. Algumas dessas funcionalidades podem ser implementadas de forma estática, como mostrado no capítulo 5.

Realizar uma implementação híbrida, com *features* estáticas e dinâmicas na mesma página, exige que o desenvolvedor entenda quais partes de página podem ser implementadas de forma estática e que podem se beneficiar dessa abordagem. Um site de *e-commerce*, por exemplo, não pode ser implementado de forma estática. Isso é porque, ao fazer buscas por produtos, o cliente força o site a gerar páginas novas de acordo com o resultado das buscas, páginas que não existiam antes no servidor, sendo, por definição, dinâmicas. Além disso, o próprio banco de dados de produtos precisa estar no servidor (seria muito inconveniente, para não dizer proibitivo, transmitir todo o banco de dados a todos os clientes), e precisa ser acessado a cada busca por produtos para que as páginas sejam geradas. Por fim, a própria efetuação da compra dos produtos no carrinho depende de comunicação com o servidor. Ou seja, a prova de conceito mostrada no capítulo 5 não se aplica a toda e qualquer página Web.

Por outro lado, uma página que implementa um questionário pode facilmente se beneficiar pela abordagem apresentada nesse trabalho. As perguntas (e respostas, caso sejam questões de múltipla escolha) podem fazer parte do próprio código da página e, portanto, estar disponíveis ao cliente sem uma requisição HTTP extra. Qualquer aplicação que precisa passar do servidor ao cliente uma quantidade pequena de dados pode fazê-lo dessa forma, sem usar uma requisição a mais apenas para esse fim. Também vimos que, caso não seja fundamental, a página não precisa ser gerada no servidor a cada requisição, mas pode já estar disponível em forma de arquivos HTML, CSS e JavaScript para ser entregue diretamente aos clientes. Isso melhora o desempenho do servidor e do site.

Um usuário pode desistir de (ou nem considerar) fazer o seu site de forma estática por desejar incluir serviços no site como seção de comentários, serviço de busca, integração com redes sociais e suporte a formulários. No entanto, recentemente, algumas empresas oferecem esses serviços integrados ao site. Basta incluir um *script* na página, que se comunica com o servidor de um terceiro que provê o serviço. Desse modo, é possível ter várias dessas funcionalidades sem precisar de um lado do servidor do site, ou seja,

a página pode ao mesmo tempo ter essas *features* e ser estática. Isso pode ter contribuído para a atual popularização de páginas estáticas e GSE.

Os GSE são ferramentas interessantes que auxiliam na criação e gerenciamento de um site estático. No capítulo 5, apresentamos a geração de uma página dinâmica com o Jekyll, um GSE. Isso é possível porque o Jekyll permite a inclusão de código JavaScript arbitrário na página (incluindo, por exemplo, um código que realize comunicação com o lado do servidor do site). No entanto, o Jekyll não permite a criação do lado do servidor do site, que precisa ser implementado separadamente. Uma reimplementação de uma página Web onde algumas de suas funcionalidades (que originalmente eram dinâmicas) são implementadas de forma estática se beneficia das vantagens de *features* estáticas: melhor desempenho e segurança. Essas *features* não estão mais sujeitas a nenhum tipo de ataque que dependa de comunicação do cliente com o servidor ou de acesso do servidor ao banco de dados.

Mas os GSE exigem que o usuário tenha pelo menos um conhecimento básico de linha de comando, além de um entendimento que o site, ao ser gerado pelo gerador, precisa, então, ser disponibilizado no servidor desejado (isso pode ser feito via FTP, o que pode, novamente, exigir conhecimento de linha de comando). Apesar de estarem surgindo algumas ferramentas que oferecem interface gráfica para alguns usos específicos de alguns GSE, essas exigências limitam consideravelmente quem pode utilizá-los. Mesmo assim, os geradores crescem em popularidade atualmente e, dependendo das páginas que se deseja criar, podem ser uma alternativa razoável aos CMS, oferecendo, inclusive, as vantagens de se desenvolver páginas de forma estática, mesmo que parcialmente, como na forma apresentada neste trabalho.

É fundamental aqui enfatizar que comportamento dinâmico é existente na Web não por pura escolha arbitrária dos desenvolvedores, mas porque esse tipo de comportamento é necessário para as mais diversas aplicações, e é responsável por usos da Internet que simplesmente não eram possíveis apenas com páginas estáticas. Também vale salientar que, apesar do comportamento estático oferecer uma melhor performance em geral e menos vulnerabilidades em potencial, isso está sujeito ao desenvolvedor. Caso uma aplicação, mesmo que dinâmica, seja implementada utilizando-se de boas práticas de programação e com ferramentas escolhidas cuidadosamente, a probabilidade de a aplicação sofrer com problemas de performance ou segurança é menor. E, no caso de um desenvolvedor utilizar a abordagem apresentada neste trabalho, os aspectos dinâmicos do comportamento da página ainda precisam ser implementados com cuidado (também os estáticos, claro).

Trabalhos futuros incluem investigar a possibilidade de gerar páginas estáticas ou com algumas funcionalidades estáticas utilizando CMS. Além disso, também seria interessante a existência de um gerador híbrido, que gera tanto o lado do cliente quando o lado do servidor de um site, e que gera de forma estática todas as funcionalidades que podem ser geradas de forma estática, se aproveitando automaticamente dos serviços de terceiros para funcionalidades não fundamentais do site.

REFERÊNCIAS

BIILMANN, M. **Why Static Site Generators Are The Next Big Thing**. 2015. Acesso em dezembro de 2017. Disponível em: <https://www.smashingmagazine.com/2015/11/modern-static-website-generators-next-big-thing>.

BUILTWITH. **CMS Usage Statistics**. Acesso em dezembro de 2017. Disponível em: <https://trends.builtwith.com/cms>.

DRUPAL. **Drupal Modules**. Acesso em dezembro de 2017. Disponível em: <https://www.drupal.org/project/project_module>.

DRUPAL. **Drupal Themes**. Acesso em dezembro de 2017. Disponível em: <https://www.drupal.org/project/project_theme>.

FONSECA, J. C. C. M. da; VIEIRA, M. P. A. A practical experience on the impact of plugins in web security. In: IEEE. **Reliable Distributed Systems (SRDS), 2014 IEEE 33rd International Symposium on**. [S.l.], 2014. p. 21–30.

GALLETTA, D. F. et al. Web site delays: How tolerant are users? **Journal of the Association for Information Systems**, v. 5, n. 1, p. 1, 2004.

JERKOVIC, H.; SINKOVIC, B. Vulnerability analysis of most popular open source content management systems with focus on wordpress and proposed integration of artificial intelligence cyber security features. **International Journal of Economics and Management Systems**, v. 2, p. 66–74, 2017.

JOHNSTON, M. **Functionality and ease of use are key factors in content management system purchases**. 2013. Acesso em dezembro de 2017. Disponível em: <https://www.cmscritic.com/functionality-and-ease-of-use-are-key-factors-in-content-management-system-purchases>.

KOSKINEN, T.; IHANTOLA, P.; KARAVIRTA, V. Quality of wordpress plug-ins: an overview of security and user ratings. In: IEEE. **Privacy, Security, Risk and Trust (PASSAT), 2012 International Conference on and 2012 International Confernece on Social Computing (SocialCom)**. [S.l.], 2012. p. 834–837.

KYAW, A. K.; SIOQUIM, F.; JOSEPH, J. Dictionary attack on wordpress: Security and forensic analysis. In: IEEE. **Information Security and Cyber Forensics (InfoSec), 2015 Second International Conference on**. [S.l.], 2015. p. 158–164.

NAH, F. F.-H. A study on tolerable waiting time: how long are web users willing to wait? **Behaviour & Information Technology**, Taylor & Francis, v. 23, n. 3, p. 153–163, 2004.

NAUGHTON, J. **A Brief History of the Future: The Origins of the Internet**. [S.l.]: Weidenfeld & Nicolson, Londres, 1999.

RINALDI, B. **Static Site Generators**. [S.l.]: O'Reilly Media, Inc., 2015.

TANENBAUM, A.; WETHERALL, D. **Computer Networks**. 5th. ed. [S.l.]: Cloth: Prentice Hall, 2011.

W3TECHS. **Usage of content management systems for websites**. Acesso em dezembro de 2017. Disponível em: <https://w3techs.com/technologies/overview/content_management/all>.

WORDPRESS. **Dashboard Screen**. Acesso em dezembro de 2017. Disponível em: <https://codex.wordpress.org/Dashboard_Screen>.

WORDPRESS. **Plugins**. Acesso em dezembro de 2017. Disponível em: <https://wordpress.org/plugins>.

WORDPRESS. **Requirements**. Acesso em dezembro de 2017. Disponível em: <https://wordpress.org/about/requirements>.