UNIVERSITY OF TARTU
Institute of Computer Science
Information Technology curriculum

Hillar Petersen

From Static and Dynamic Websites to Static Site Generators

Bachelor's thesis (6 ECTS)

Supervisors: Karl Blum, MSc
Tõnu Tamme, MSc

Tartu 2016

From Static and Dynamic Websites to Static Site Generators

Abstract:

Dynamic Content Management Systems like WordPress are used on almost half of the world's active websites. As many of these sites are content-driven, like blogs, news sites, personal, company and organisation websites, rendering them dynamically does not offer any value compared to if they were static.

Paradigmatic differences between static and dynamic websites are analysed and the benefits of each described. It is found that for static-by-nature websites, static approach has core benefits such as security and end-user performance, as benefits of dynamic platforms come mainly from the more mature toolset.

Features and usability of three popular Static Site Generators – Jekyll, Hexo and Hugo are analysed. It is found that Jekyll and Hugo are more suitable for universal websites, as Hexo is oriented for blogging. Hugo should be preferred for a large website, as its site generation speed is significantly faster than Jekyll's. However, the extensibility of Hugo is more complicated.

Additional tools in the growing static websites ecosystem are pointed out. Some ways of combining these to create a complete toolset are given and ideas for future development proposed.

Keywords:

Web development, Content Management System, Static Site Generator, Jekyll, Hexo, Hugo

CERCS:

P170 Computer science, numerical analysis, systems, control

Staatilistest ja dünaamilistest veebilehtedest staatiliste veebigeneraatoriteni

Lühikokkuvõte:

Dünaamilised sisuhaldustarkvara paketid, näiteks WordPress, on kasutusel peaaegu pooltel maailma aktiivsetest veebilehtedest. Paljudel neist lehtedest on peamiselt eelloodud sisu – näiteks blogiartiklid, uudised ja isiklikud või ettevõtete veebilehed. Sellise iseloomuga veebilehtede esitamine läbi igakordse dünaamilise genereerimise ei lisa mingit väärtust võrreldes sellega, kui lehed oleksid eelgenereeritud ehk staatilised.

Käesolevas töös on analüüsitud staatiliste ja dünaamiliste veebilehtede põhimõttelisi erinevusi. On leitud, et staatilised lehed omavad sisulisi eeliseid dünaamiliste ees – näiteks turvalisus ja kiirus – kuid dünaamiliste veebilehtede eelised seisnevad peamiselt küpsemate tööriistade olemasolus.

Töös on võrreldud kolme populaarseimat staatilise veebilehe generaatorit – Jekyll'i, Hexo't ja Hugo't. On leitud, et Hexo sobib hästi blogimiseks, kuid Jekyll ja Hugo ka universaalsete veebilehtede loomiseks. Hugo't tasub eelistada suurte veebilehtede puhul tänu selle oluliselt suuremale genereerimiskiirusele, kuid peab arvestama selle keerulisema laiendatavusega.

Staatiliste veebilehtede ökosüsteemi on põgusalt tutvustatud ning toodud välja vahendeid lehtede majutamiseks, graafilisi kasutajaliideseid jmt. On pakutud ideid, mida tasuks staatiliste veebilehtede tööriistades edasi arendada.

Võtmesõnad:

Veebiarendus, sisuhaldustarkvara, staatiline veebigeneraator, Jekyll, Hexo, Hugo

CERCS:

P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Contents

Introduction

In the world there are more than billion websites. These sites are served by more than six million computers. Tens of millions new websites are added each year. These are large numbers so it matters to do it right.

Why? By choosing the optimal approach and tools we can save energy to host these sites – thus reduce the cost of hosting and make our contribution to slow down global warming. By using the right tools to develop sites, web developers and companies can save time and effort – thus reduce the cost of development which will hand them a competitive edge. Needless to say, website owners would win from higher productivity as that means they can get their sites launched faster (or get a superior site in the same time) and could mean less development costs.

In general, there are two types of websites – static and dynamic ones. Static is the "good old" that has made a comeback thanks to static site generators. New tools are created every day that make the static approach continuously more attractive. At the same time dynamic sites still have their advantages and many dynamic website tools are also improving.

In this quickly evolving web world and with a plethora of tools available it is tough to understand the situation and make educated decisions which approach to take or which tools to use. Performance, easy scalability, less vulnerabilities and lower cost of hosting are key factors why static is gaining ground.

The main goal of this thesis is to analyse the current situation and tools for creating, managing and hosting static websites, and propose ideas for moving forward.

This study is divided into four main parts. The first part describes the basic terms, background information and defines the problem in current situation. The second part of this thesis further compares the differences between static and dynamic approaches and current state-of-the-art tools. The third part gives a deeper overview of currently available static site generators and how they can be coupled with other currently available tools to support the whole website development and management lifecycle. The fourth part looks into the near future where the best parts of static and dynamic approaches could be combined and the author of this thesis proposes key points for the development of a potential next generation static solution with some modern extras.

1. Background

Websites are something we use every day when we read news, search for information, watch videos, pay our bills and follow our favourite bloggers. Websites present information and offer services. Technically, a website is accessed over HTTP or HTTPS protocols, identified by a URL (often with a "www" in its address) and consists of a set of linked documents which a web browser presents to the user [1].

As of July 2016 there were [2] [3]:

- over 325 million registered domains
- provided by more than 6 million computers
- serving over 1 billion websites

However, Netcraft estimates that less than 20% of these sites are active or *real* sites – the other 80% are computer-generated sites not offering unique content, such as parked domain websites displaying ads[1]. This makes the current number of active websites around 170 million.

These numbers indicate publicly (Internet) accessible websites – as there is no data about the number of websites accessible in private networks only (such as organization intranets) but we can assume there are many of these as well.

1.1 Types of websites

Static

The original website type [4]. In the current study an expanded definition is used – a static site consists of HTML pages, documents and media which can be read by the server from persistent storage and served to the client (usually a web browser) without further customization, as is illustrated by Figure 1. No runtime changes or other operations are done to the stored page by the server. Static website can include HTML, CSS, multimedia content and possibly client-side scripts/programs written in Javascript or other programming languages that the browser is able to run. These sites are usually content-driven (read only), but can be interactive like a game or a tool (e.g. a calculator[2]).

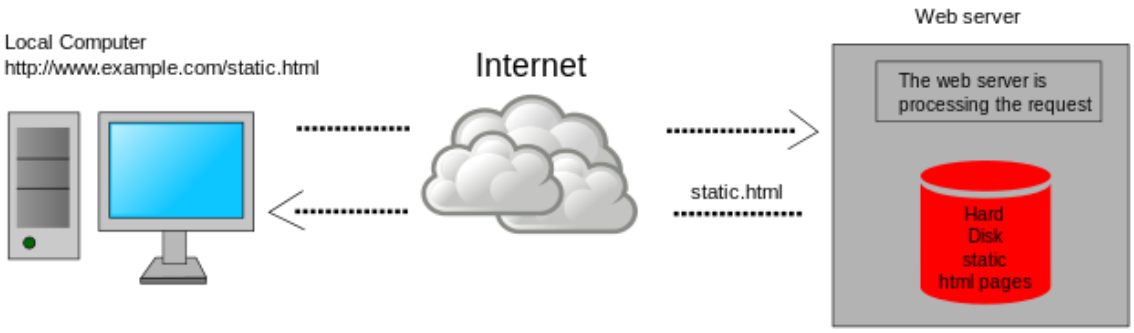

Figure 1. An example how browser requests and server responds with a static page. [5]

[1] https://en.wikipedia.org/wiki/Domain_parking
[2] http://www.a-calculator.com/basic/embed.html

Dynamic

Websites that are stored in a format which is not directly renderable by the web browser, are called dynamic sites. The server takes client request and generates a response that is renderable by the browser after performing some actions in between. Meanwhile operations are defined in a scripting or programming language (some of the most common for that being PHP, Python, Perl, Ruby, Java, C#, JavaScript via Node.js) and often include database requests, as illustrated by Figure 2. Site can be naturally dynamic, providing interactive client-server functionality or it can also simply convert statically stored information to a HTML page.

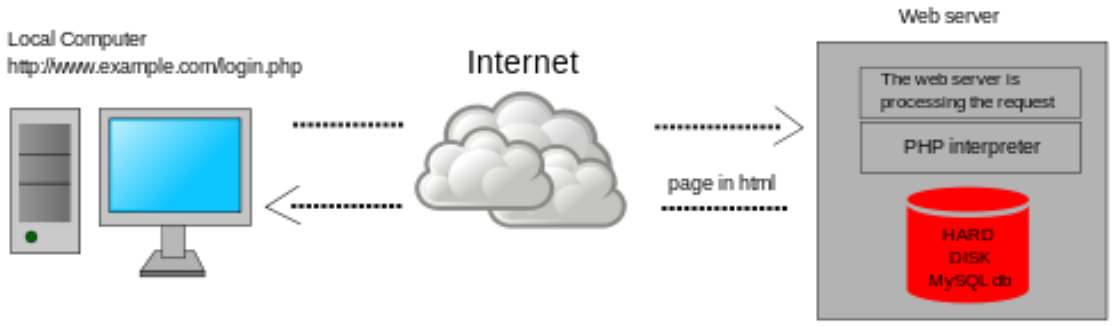

Figure 2. Server gets HTML through PHP interpreter which uses MySQL database [6].

1.2 Website components and lifecycle

Here website means a website accessible via Internet unless specified otherwise.

Basic components

The main components for websites are:

- **The site itself.** Either static HTML documents or a dynamic script/application/CMS site, with possible assets (mostly CSS, Javascript files, images and other multimedia content).
- **HTTP server**[3]. Takes the client's request, processes it (in case of a dynamic site proxies the request on to a script/application or an application server) and returns the HTML page with its assets. The HTTP server returns the site from a specific URL – an address where the site can be requested from. URL usually includes a human-friendly **domain name** which is resolved to the HTTP server's IP address by the Domain Name System[4].

Dynamic sites can include:

- **Database.** Used to store most of the dynamic website data (most of the configuration and content of a site and can include user-specific data). Usually a relational open-source database like MySQL, MariaDB or PostgreSQL is used. As possible future technologies of dynamic CMSs, alternatives could be explored – like document-based MongoDB which might be well-suited for storing web content, as websites consist of documents.

[3] https://en.wikipedia.org/wiki/Web_server
[4] https://en.wikipedia.org/wiki/Domain_Name_System

- **Template engine.** Parses web templates to output a HTML webpage, possibly injecting data bits read from database and combining template partials (often reusable/common parts of pages like a header, menu or footer). A dynamic site utilizes a template engine runtime (might use preprocessors to boost performance), or in case of a static site, a template engine[5] might be used during site generation process.
- **Themes.** Theme is a package containing graphical appearance details. In case of a webpage it is made of templates that define layout (HTML, CSS) and stylesheets (CSS) along with the graphical assets[6].
- **Plugins.** Extensions to the core dynamic platform. On open source or open plugin architectures plugins can be developed by anyone. For instance WordPress has more than 45 000 plugins [7].

Both static and dynamic sites can utilise:

- **Admin portal (Content Management System – CMS)[7]**. In dynamic solutions CMS usually refers to both admin portal and core in a single package. An integral part of dynamic websites – is used to manage the structure of the site, page layouts, themes, plugins, content, metadata and other details, configure settings, users, publishing workflow etc. For dynamic sites the data is kept in storage as partials which mostly persist in database and several files in the filesystem. However, as a newer trend, there are also admin portals which generate static output – this will be opened later in this study.
- **Hosting platform**. A platform with server resources and a set of tools to build, deploy, and host a site. Possibly can provide a hosted version of a dynamic CMS.
- **Content Delivery Network (CDN)**. CDN in the context of websites is a globally distributed network of reverse proxy servers deployed in multiple data centres [8]. A CDN stores whole static sites (HTML pages with their assets) concurrently in many places of the world and with the help of DNS serves the site from the "best" server (generally speaking from the server closest to the client). On dynamic sites CDN can cache static assets (files like .css, .js, images, videos) even if the page generation is dynamic.

Static sites can be generated via a **Static Site Generator (SSG)**. A static site generator is a template engine that outputs static assets—HTML, CSS, and JavaScript. It takes the content, typically stored in flat files rather than databases, applies it against layouts or templates and generates a structure of purely static HTML files that are ready to be delivered to the users [9].

Lifecycle

By the author's experience with web-based software development projects, at least three skillsets are needed for professional web sites: web designer, web developer and content editor (for web developers, content editor can usually be named "the client"). Some usual technical outputs of each role are:

- Web designer works with themes and basic templates
- Web developer creates a repository, creates the site core, installs server and development environments, creates build procedure, includes plugins/external modules, creates custom frontend and backend code as needed, possibly creates site structure

[5] https://en.wikipedia.org/wiki/Web_template_system
[6] https://en.wikipedia.org/wiki/Theme_(computing)
[7] https://en.wikipedia.org/wiki/Web_content_management_system

- Content editor creates content pages and adds content along the lifecycle

Lifecycle of a website might go like this

1. A website is first developed in a local development environment. (At the same time content editors might create content with alternative tools like a word processor – as the web environment is not yet available.)
2. The designer creates templates, stylesheets etc.
3. At some point the site core and basic structure elements have been created by the developer and at least some of the designer's templates included. Then either a server-side content creation environment is established (with an admin interface and possibly password-protected before an official launch) or an alternative is provided to content editors for editing and reviewing the content (like a local option with git integration as with SSGs).
4. As the site and content is evolving, at some point it will all be reviewed and launched to production environment (usually via a continuous deployment process, but can be manual copying, e.g. via FTP). For a smaller website and when using a dynamic platform, from there on production will be the only environment for both content editing and for the end-users, possibly with a content preview in admin interface (or there can be a local editing possibility like with a SSG or even via a mobile app).
5. From there on, content editor occasionally adds new content or modifies the existing one. On content-driven sites the content editor usually adds also new pages and subpages which can be launched independently of the code. The developer might continue to develop code and occasionally launch.

1.3 Main technologies and tools

HTTP servers

The main servers used to host active websites are Apache Web Server with 46%, Nginx with 22%, Microsoft IIS with 10% and Google Web Server with 9% of sites. Nginx is the only one of the top servers that is constantly gaining market share [3]. There are good reasons why Nginx is getting more popular, among these the ability to serve static content more efficiently than Apache [10].

HTTPS is now going mainstream, as free (and automated) SSL certification is possible via Let's Encrypt.

Clients

Humans access websites from their computers (desktops/laptops) and mobile devices (tablet/smartphone) via a browser. Global statistics according to StatCounter as of July 2016 [11] state that of all page views about 50% were made from computers and 50% from mobile devices (mostly smartphones) with the trend insisting that by the time when the current study is published mobile has already overtaken desktop. The most popular browser is Chrome with about 50% market share, followed by Safari (the browser on Apple devices) with 18%.

Static solutions

Some stats will be presented as found on staticsitegenerators.net, staticgen.com and W3Techs.com CMS popularity rankings [12].

There are 437 static site generators at the time of writing (with the count steadily rising), with Jekyll[8], Hexo[9], Hugo[10] and Octopress as the most popular ones based on the number of stars in GitHub[11]. According to W3Techs, static tools GitHub Pages[12], Jekyll and Hugo have also been listed as CMSs, with GitHub Pages having 0.3% market share (26th place) among known CMSs and the rest less than 0.01%. That would mean GitHub Pages is used as CMS for over 200 000 sites. For the rest, it must be stated that as the static templates are fully under developer's control there might not be information in page HTML source to correctly identify whether and which SSG was used to generate a page – as such W3Techs numbers about SSGs might be smaller than actual.

Today, static site generators are the new standard for creating static web sites even though in the most basic case any text editor like Sublime Text would do (a text editor is needed for using SSGs as well – likewise for creating customized dynamic sites). In fact even rich text processing software like Microsoft Word can be used to output HTML. However, we will concentrate on SSGs as these are the new tools that could compete with more mature dynamic site creation tools.

Static site hosting: any regular web hosting provider or your own web server will do, but API-accessible *smart hosting* provider is the new trend – for instance GitHub Pages and Netlify[13]. A number of sites hosted on GitHub Pages have also been created by a SSG like Jekyll and not by the Automatic Site Generator[14] and as such the number of sites hosted on GitHub Pages *should* be larger than the number of sites using GitHub Pages as a CMS.

Dynamic solutions

Dynamic websites are based mostly on one of the popular CMSs – or can be built on a custom CMS. The latter might be reasonable for large sites needing lots of custom functionality – which might be more productive to be built not having to depend on an existing platform and not having to maintain compatibility with the original platform for future version upgrades or security patches.

According to W3Techs.com the most popular CMS is WordPress (WP)[15] running 26% of all active websites (59% of all sites built with an identified CMS). WP is followed by Joomla and Drupal which have ten times less market share – 2.7% and 2.2% respectively.

A note about accuracy of W3Techs's static vs. dynamic CMS stats. Around 55% of active websites are listed as *not being based on a known CMS* [12]. One can roughly assume most of these sites are hosted statically. Although W3Techs includes some static tools in their content management systems list, like HTML editors Adobe Dreamweaver and Microsoft FrontPage, static CMSs GitHub Pages and Movable Type[16], their market shares are less than 1%. At the same time some of the dynamic CMSs are not identified by W3Techs statistics, like an Estonian CMS Voog (previously known as Edicy), which claims to host "hundreds of thousands of websites"[17]. As such, Voog and possible other unlisted dynamic

[8] https://jekyllrb.com
[9] https://hexo.io
[10] https://gohugo.io
[11] https://github.com
[12] https://help.github.com/articles/what-is-github-pages/
[13] https://netlify.com
[14] https://help.github.com/articles/creating-pages-with-the-automatic-generator/
[15] http://wordpress.org
[16] https://en.wikipedia.org/wiki/Movable_Type
[17] http://www.voog.com/about (8th Aug 2016)

tools (like custom CMSs – which mostly power larger sites and thus there are not too many of these) can equal out the share of some static tools listed.

Hosting providers and platforms

According to Netcraft among the world's most popular hosting providers are GoDaddy[18], DigitalOcean[19] and OVH[20], while Amazon has the most servers and IP addresses.

GoDaddy offers limited CDN support for serving static assets (for US-based sites only, and with other limitations). DigitalOcean does not offer a CDN by default, so a custom configuration must be used to enable it[21] and an external CDN provider will cost some extra. Similar configuration would work with GoDaddy.

OVH provides different levels of their own CDN support as part of their hosting packages, WordPress and Joomla are preinstalled.

WordPress.com, the hosted version of WordPress, hosts over 100 million blogs and can be considered the largest WordPress blog hosting site [13] [14].

Large hosting providers usually focus on VPS or shared hosting packages for websites, with domain name registration, website hosting, data storage and e-mail. WordPress is usually installable in one click, WordPress instances can be cloned etc. These packages usually cost only a couple of euros per month (standard packages vary from €0.99 to about €10/month). Better providers also offer API support to utilize the services automatically via scripts.

Some alternative smart static hosting platforms are: GitHub Pages, Netlify, Aerobatic[22].

There are also modern cloud-based services (e.g. Amazon AWS) which also allow hosting of websites through a custom build process, provide APIs to work with them automatically, and many provide a CDN as well.

CDN providers

W3Techs statistics show that a CDN is used on less than 7% of websites [15]. CloudFlare[23] powers 70% of the CDN-using sites (5% of all websites), followed by Akamai[24] with 12% and Incapsula[25] with less than 6%.

GitHub Pages uses Fastly CDN[26] as a reverse proxy for all of their incoming traffic.

Netlify uses their own CDN for HTML pages and Akamai CDN for assets [16].

Behind the scenes, reverse proxy servers Apache Traffic Server, Nginx, and Varnish power many CDNs [17][[18]] and can be used to build a custom CDN as well.

Examples of websites and used technologies

Landing pages for many popular sites are static, like apple.com, healthcare.gov, github.com and mailchimp.com. These sites are using API-based JavaScript calls to

[18] https://godaddy.com
[19] https://digitalocean.com
[20] https://www.ovh.ie/
[21] https://www.keycdn.com/support/digitalocean-cdn-integration/
[22] https://aerobatic.com
[23] https://en.wikipedia.org/wiki/CloudFlare
[24] https://en.wikipedia.org/wiki/Akamai_Technologies
[25] https://en.wikipedia.org/wiki/Incapsula
[26] https://www.fastly.com/customers/github

backend services when navigating deeper or calling functions like search from the homepage. GitHub.com is hosted on static hosting service GitHub Pages. Mailchimp.com, which is 369th of the world's most visited websites, uses Akamai CDN and Nginx web server and according to middlemanapp.com, is built using Middleman SSG.

Many well-known sites run on dynamic CMSs – some examples include nytimes.com, newyorker.com, fortune.com and nationalgeographic.nl running on WordPress. It can be seen that all those sites are mainly content-driven – and could run on a static platform with some non-static additions. The same goes with University of Tartu website ut.ee which is running on Drupal[27].

Some websites generated by a SSG are: mpc-hc.org (Jekyll, Apache) and gohugo.io (Hugo, Nginx web server, CloudFlare CDN).

The busiest sites[28] use a CDN regardless of whether they are on static or dynamic platforms, like can be seen through W3Techs Site Info tool[29], Intricately[30] or looking at the pages' HTML sources. For instance, google.com uses Google CDN, youtube.com combines Akamai and Google CDN and facebook.com uses Akamai CDN.

Web applications

There is a specific subset of dynamic websites which we will not analyse in the current study – these are functionality-driven web applications whose main purpose is to interact and behave based on user-specific data[31]. These applications often include significant custom development on server side, require user authentication and do numerous read-write operations on datastore. Examples of web applications are all kinds of custom functional solutions, non-trivial e-shops, self-services, social networks – like facebook.com, amazon.com or energia.ee after logging in. These are not in our scope as these would be quite difficult to build using SSGs. Although as a sidenote Single-Page Applications[32] are becoming the best practice architecture of web applications which have API-based backends and separate frontends thick on client-side code – thus have some similarities to modern static solutions that might have a few "dynamic" extras and are making API calls via JavaScript. Modern web apps built with Ember, AngularJS, React or Aurelia can be deployed as static websites and served directly from a CDN with API backend shared, for instance, between the website's UI and the mobile client. However as these apps are not normally generated with a SSG and they contain a principal amount of custom logic, their qualities depend very much on the specific implementation – as such they are not relevant in this study's more general static-dynamic discussion and static tools comparison. Here they are called web applications, not websites.

A sidenote. More detailed reading on core web technologies can be found via Cody Lindley's Front-end Handbook[33]. That book is created with a popular Static Site Generator optimized for generating documentation sites and e-books, GitBook[34].

[27] https://w3techs.com/sites/info/ut.ee
[28] http://www.alexa.com/topsites
[29] https://w3techs.com/sites
[30] http://my.intricately.com/
[31] https://en.wikipedia.org/wiki/Web_application
[32] https://en.wikipedia.org/wiki/Single-page_application
[33] http://www.frontendhandbook.com/practice/tech-employed-by-fd.html
[34] https://gitbook.com

1.4 The problem

There are billions of blog posts [19], news articles, institutional/company and personal webpages – and other content which is hosted dynamically. Yet these content-driven sites do not actually offer any (significant) dynamic functionality to the visitor. In this study these sites will be referred to as "**static-by-nature websites**". These sites mainly present content, but may include minor dynamic features like form submission (e.g. comments or a contact/registration form). The author's opinion is that these sites have been created dynamically for the reason that for years there have been convenient dynamic site tools around (dominantly: WordPress) but not static site ones. The browsers now support running thick JavaScript applications and HTML5, offering many possibilities which were previously reasonable to be implemented only on server side. All kinds of API-based JavaScript widgets are available – things like commenting, posting to social media, search, real-time data updates and even simple e-shops are possible to add to a static website.

A hypothesis is that static-by-nature sites would benefit if they actually *were* static. It will be analyzed in this study (see "Comparison of static and dynamic approaches").

Both static and dynamic tools have their advantages and disadvantages. For instance, static-by-nature sites which are hosted on dynamic platforms are technically more complex, consume more computing resources and are unnecessarily costly – compared to if they were hosted statically. Static sites have limited functionality. As the modern static approach has potential to replace many currently dynamically solved use cases (i.e., static-by-nature websites), but the toolset is relatively new, some most popular existing static tools will be looked into in this study. SSGs will be compared, some new tools identified and potential overall solutions studied.

As basis for this study no suitable academic resources were found. There are several blog posts discussing static vs dynamic approaches – for instance, M. Biilmann's post "Why Static Website Generators Are The Next Big Thing", in Smashing Magazine [20]. Likewise, there are posts where static site generators are compared, for instance, M. Biilmann's post "Static Website Generators Reviewed: Jekyll, Middleman, Roots, Hugo" in Smashing Magazine [21].

Results of the study will hopefully be useful for web developers and website owners choosing their next website platform – and propose ideas for future static platform developers.

2. Comparison of static and dynamic approaches and tools

Comparison will be broken down into two categories – one from end-user's perspective and the other from site administrator's perspective.

2.1 End-user experience

Benefits of static solutions

Though developers give many reasons why they prefer static sites (some of which will be referenced to in the current study) most of them can be summarized as the holy grail of end-user solutions: better user experience (UX)[35]. Parts of user experience are specific non-functional requirements (NFRs)[36] which describe the expected system quality.

While end-user experience consists of a myriad of components and due to psychological, cultural and other quite ambiguous factors there can be no single and straightforward definition for all of UX, we can point out some NFRs giving a relevant contribution:

- **Page load time**. There are many good reasons why fast page load time is important – better user experience and impact on search engine rankings among them [22]. The importance of fast load time is enormous – 40% of users abandon a website if it loads longer than 3 seconds [23]. During a single page load quite a lot happens in a dynamic site (for instance in WordPress[37] [38]) while in a static case the HTTP server will return the stored page from disk or cache without any additional overhead. If a site would be hosted dynamically and statically with similar infrastructure and HTTP server configuration then in the best case load time of a page in the dynamic version can be *close* to the static one, but never better – as part of the components are the same for both solutions plus a dynamic site has additional components (e.g. a scripting language interpreter and a relational database). For performing close to static solutions dynamic ones need quite a bit of configuration work and additional components or plugins (for instance, see WordPress optimization guide[39] and how WP Super Cache plugin works – making pages as static as possible[40]).
- **Efficient scalability**. Simply this means cost-effective readiness for serving more and more visitors as the website gets more popular, without compromising page load speed. Scaling dynamic sites can be a quite complex task as there more underlying components (e.g. CMS and database) which need to be scaled. Especially complex when you need to scale in short time – like during a quickly peaking load. Even the simplest setup for scaling a dynamic CMS easily is far from simple and low-cost[41]. Scaling a static site via a CDN is already there and the whole site can be cached, not just static assets. Even if a CDN is not used, simply adding web servers and using a suitable load balancing method[42] will be sufficient for most static sites.

[35] https://en.wikipedia.org/wiki/User_experience
[36] https://en.wikipedia.org/wiki/Non-functional_requirement
[37] https://www.rarst.net/wordpress/wordpress-core-load/
[38] https://codex.wordpress.org/Plugin_API/Action_Reference
[39] https://codex.wordpress.org/WordPress_Optimization/Caching
[40] http://nilclass.com/courses/what-is-a-static-website/#13
[41] https://cloudonaut.io/wordpress-on-aws-you-are-holding-it-wrong/
[42] https://www.nginx.com/blog/scaling-web-applications-nginx-part-load-balancing/

- **Availability and security** of the service, meaning, for instance, that the service is not unresponsive, does not have its content modified by hostile hacking and does not allow users' data to be accessed by unauthorized parties. Although lots of plugins to choose from allow to boost productivity of dynamic site development, probably most of WordPress's 45 000 plugins have vulnerabilities [24] (which do not impact their popularity numbers [25]). Millions of WP sites can already be compromised. In the worst case, the dynamic site owner could suffer a major data breach, like Mossack Fonseca earlier this year[43]. And dynamic sites which are not patched regularly risk to help spread malware acting as bots, like happened with Drupal 7 in 2014[44] and is probably the case with millions of WP sites as well, without the administrator even knowing it. Static sites are not affected by the same problem, as their server-side security is much easier to maintain thanks to using only client-side technologies and not opening possibly vulnerable server-side services.

CDN brings additional benefits:

- **Lower latency of sites with an international user base** (hosting close to users' physical location). In a test by Mathias Biilmann [21], Smashing Magazine website[45] (running on WP in a single data center) was fetched and deployed statically to Netlify (cached in CDN). Single-user tests to compare page load time from different locations of the world with Sucuri[46] were made. On average, static version in CDN returned pages 6 times faster. As another example, ut.ee which is hosted in Tartu, has a reasonable page load time from Europe, but quite poor for visitors from US, Asia or Australia[47].
- **World-wide replication and failover mechanisms provide availability** – e.g., if a CDN server in Stockholm is "closest" to the client, content will be served from there. But if the Stockholm server has availability issues requests can be automatically redirected to a server in Warsaw[48]. CDNs are reported to have stopped large DDoS attacks successfully [26]. CDNs, like Google Cloud CDN can interconnect[49] with each other with CDNI[50], emphasizing on their benefits even more.

Other factors:

- Consuming less computing resources means less power consumption (means less cost for hosting and **less negative impact on the environment**). Even if many data centres might use only half of their energy consumption on actual server resources [27] saving an order of magnitude computing time could make an important impact. (Exactly how much energy could static sites save compared to dynamic ones will need to be studied separately.)

Benefits of dynamic solutions

Natural benefit of dynamic solutions is **write time**, i.e. if any changes are done to the page (like new content added or site structure changed) then with dynamic solutions the new

[43] https://www.wordfence.com/blog/2016/04/mossack-fonseca-breach-vulnerable-slider-revolution/
[44] https://nakedsecurity.sophos.com/2014/10/30/millions-of-drupal-websites-at-risk-from-failure-to-patch/
[45] https://www.smashingmagazine.com
[46] https://performance.sucuri.net/
[47] https://performance.sucuri.net/domain/ut.ee (10th Aug 2016, rating: F)
[48] https://www.cloudflare.com/features-cdn/].
[49] https://cloud.google.com/interconnect/cdn-interconnect
[50] https://en.wikipedia.org/wiki/Content_delivery_network_interconnection

live page can be seen by users almost immediately (maybe a couple of seconds are needed for writing changes into database). Only the changed part is written into database and can be seen by a user accessing the site immediately after that. For static solutions, larger sites can be generated for tens of seconds, plus they must be uploaded to the hosting provider. If a CDN used, long cache invalidation time of HTML files might mean an additional delay for updating static sites or seeing an impartially updated state of the site [16].

Real-time or personalized content is native as each page is rendered based on the current data exactly when requested. With static sites, features must be added via JavaScript and likely a CORS-supporting API based on an external backend.

2.2 Site owner's experience

From site owner's perspective, the first and most important is end-user's experience, described in the previous chapter.

Principal actions for creating and maintaining websites are creation of the site core, structure, look & feel, create/add/edit content, review and publish to live site.

Main roles included are web developer, web designer and content editor – differentiated by the according skillsets. Possibly, a person (like a personal site owner) could fill all of these roles with a suitable match of tools and skills, and in a large project there may be several people or even teams of each.

Necessary skills for creating a website with a SSG and with a Dynamic CMS can be illustrated by Table 1.

Table 1. SSG vs. dynamic CMS – actions and roles

Role	**Legend**
Dev – developer	Without () – the most natural role and action fit
Designer – web designer	() – can be done by in some cases or help needed from
CE – content editor	(()) – can be done by in rare cases or very little help needed from

	Create site core	**Create look & feel**	**Create & update site structure**	**Create & update content, review, publish**
SSG	Dev	Designer ((Dev))	Dev ((CE))	CE + Dev
Dynamic CMS	Dev (CE)	Designer (Dev) (CE)	CE	CE

Create site core. With a hosted dynamic CMS, a non-technical content admin could do everything to set up a simple site themselves, via a graphical admin interface. Static tools are not quite there yet.

Create and update look and feel. The process for professional websites includes having a web designer first design the page in a graphic design software like Adobe Photoshop or Illustrator. Then the graphical presentation needs to be implemented in HTML, CSS and

images – meaning the creation of layout templates, styles and graphical assets. SSG templates are often simpler than dynamic CMS templates as they do not include back-end calls written in a scripting language. As such, when creating SSG templates, designer's need for developer's help is smaller – if at all. However, the most popular dynamic CMSs can currently benefit from a large template and theme base which can be used by content editors directly and somewhat customized via a graphical user interface (e.g. in WordPress). In dynamic content management systems theme selection or theme change of the website often takes just a few clicks and the admin can choose between many pre-defined themes in order to easily change the look and feel.

Create and update site structure. This means creating and updating site navigation hierarchy (pages, subpages etc.). With a SSG this logic is created and updated via not-so-easy to understand plain text files which is good for developers, but not that good for content editors.

Create and update content, review, publish. Adding blog/news posts and working with other content pages, testing/reviewing the results and updating production site. With SSGs pre-defined types of content can be created by content editors in a reasonable way. Convenient means are provided for the editor to review the changes in a local environments (on editor's computer via a browser), but it is more difficult for a collaborative process as at least a git pull request would be needed and there is no central staging environment. Production update (publishing) mostly needs some technical skills. Later in this study will be presented more details on differences between SSGs and possible new additions.

All in all, looking at the table, SSGs are quite developer-centric. "Developer skills" are needed in order to work with SSG-based sites and the "normal" site owner or content editor cannot conveniently manage the website on their own. However – developers with a suitable custom toolset will likely be able to pull off a more efficient process with static tools, as git, build tools incl. asset pipelines etc. are used by default.

In dynamic CMSs, basic **dynamic user-input features** are available out-of-the-box and configurable even by non-technical staff, like commenting and form submission. However, there are already easy-to-integrate tools to achieve similar functionalities with static sites [9] and likely many new coming.

Another benefit of dynamic CMSs is that the existing **toolset is richer**, with many CMSs supporting:

- full process from development to content editing to publishing
- graphical admin user interfaces (translated to many languages)
- lots of features and plugins
- abundance of ready-made templates/themes
- large number of developers
- abundant support from both community and hosting providers
- multi-site support for managing many sites from a single interface

3. Comparison of tools for creating and editing static websites

A comparison of the features of the most popular static and dynamic website creation and management tools (SSGs) will be given. Some possible interesting alternatives will be pointed out. A few possibilities to combine different tools will be described.

3.1 Static Site Generators

In short – during 26 years static site generation tools have evolved from Tim Berners-Lee's first web browser, then text editors, then HTML editors, then a few dynamic CMSs capable of producing static output (e.g. Movable Type[51]), and now to SSGs.

Concentrating on SSGs, there are hundreds of them, all vying for adoption, written in Ruby, Node, Go, Python, JavaScript and other languages. Most of them share similar traits [21]:

- **Templating.** Allowing a website to be split into layouts and includes to get rid of repetition is one of the basics of static website generators. This means a page is based on a reusable layout which includes reusable partials (e.g. page metadata, a menu, header, footer, sidebar, content items, also local variables), content is kept in separate files. Different templating languages/engines are supported by different SSGs – for instance Liquid, Swig, EJS, Haml, Jade.
- **Markdown support** (or some other markup language). Markdown[52] is easy to work with, and several Markdown editors are available, like StackEdit[53]. Markdown is supported by programmer-oriented text editors as well. Alternatives include reStructuredText, Textile and others[54] – as supported by different SSGs. In general, they all allow content developers to write content in a structured format plain-text files.
- **Front matter** is the bit of meta data, typically in YAML format, at the very top of a document, e.g.:

  ```
  title: Title of the document

  tags: [tag 0, tag 1, tag 2]

  ---

  # Actual content

  Here starts the content.
  ```

  The example front matter data would be shown as post title and tags on the published post. This makes annotation of single-file documents with metadata straightforward and have data in a human-readable text format rather than stored in a much more opaque format in a database.

[51] https://movabletype.org/documentation/administrator/publishing/static-and-dynamic-publishing.html
[52] https://en.wikipedia.org/wiki/Markdown
[53] http//stackedit.io
[54] https://en.wikipedia.org/wiki/Comparison_of_document_markup_languages

- **Asset pipeline.** Front-end development today involves several build tools and compilers. Before going to production it is useful to minify and bundle assets. CSS preprocessors like Sass, SCSS, Less and Stylus are mainstream tools. CoffeeScript, TypeScript and ECMAScript 6 transpiling have made JS compilers a part of programming for the browser. Browserify allows developers to focus on building structured JavaScript without worrying about bundling. Developers might want to lint their code with ES6Lint. Most modern static website generators include an asset pipeline that handles asset compilation, transpiling, minification and bundling. Some SSGs are based on build tools such as Grunt, Gulp and Broccoli, and let site generation hook into a larger ecosystem of tasks and build steps, allowing web developers to use any pipeline. Immediate browser refreshing when a file is saved has also become standard for many generators.
- **Output is a complete static website** which can be hosted independently of the SSG in any static hosting solution.

Content publishing process. A typical use case of changing content with SSG is writing a blog post or adding a news item, which could go like that:

1. New post is composed in a text editor and formatted in Markdown – and committed to git repository.
2. Page is regenerated – most SSGs have a watch functionality to detect file changes and regenerate the site automatically.
3. Site is previewed locally – many static generators provide their own local server to add some development features like automatic page reloading upon file save.
4. Changes are pushed, site is regenerated (built) for production and deployed to a live server (regular hosting or CDN) – either via custom scripts (e.g., as a task in Grunt), automatically by a smart hosting service watching Git repository, like GitHub Pages or Netlify [28], or by copying files to a hosting location manually.

Comparison of the most popular SSGs – Jekyll, Hexo and Hugo

Jekyll, Hexo, Hugo are in active development – at the time of writing all have been updated just a few hours ago. They are are open-source, with MIT and Apache 2.0 licenses, as listed by staticsitegenerators.net. In this study a high-level comparison will be done – this is reasonable as SSGs are in constant change and this study is not focusing on specific use cases, but rather on principles.

Hexo calls itself blog-oriented, as Jekyll and especially Hugo are more general SSGs. Comparison details can be found in Table 2.

Table 2. Comparison of Jekyll, Hexo and Hugo

	Jekyll	**Hexo**	**Hugo**
Programming language	Ruby	JavaScript (Node.js)	Go
Templating languages	Liquid	EJS, Swig[55]	Go Templates
Content markups	Markdown or HTML (Textile, Slim, Jade, Pug, HAML, reStruc-	Markdown (pandoc, Textile, reStruc-	Markdown (AsciiDoc, reStructuredText via

[55] more are supported via plugins

	turedText, AsciiDoc via plugins)	turedText via plugins)	external helpers)
Admin or editing interface with a GUI	No (supporting packages for text editors, e.g. sublime-jekyll)[56][57]	Hexo-admin plugin (blog-oriented Markdown editor)	Rango and Hugopit as Tools), supporting package for Sublime Text
Preview mechanism	Local server, auto regeneration (live reload via generator-jekyllized/Browsersync server or Hawkins/LiveReload browser extension)	Local server, auto regeneration (live reload possible via hexo-livereload and hexo-browsersync plugins)	Local server, auto regeneration, live reload
Custom content types, data-driven content	Yes (collections)	No	Yes
Custom sections, custom includes	No	No	Yes
i18n support*	Yes (via i18n filter or jekyll-i18n_tags plugin)	Yes	Yes
Built-in asset pipeline	Yes	Yes	No
Atomic regeneration	Yes (--incremental flag, stated as experimental)	No	No
Site generation time	Slow (fast with --incremental)	Fast	Very Fast
Deployment support	Git[58], S3, Rsync, possibly more	Git, Heroku, OpenShift, Rsync, FTP, S3, CloudFront via plugins	Manual configuration (via Wercker[59]) Amazon Lambda support and SFTP (via tools)
Migration tools	>=20 platforms[60]	From RSS, Jekyll, Octopress, WordPress, Joomla	From Jekyll, WordPress, Ghost, OctoPress, Tumblr, Drupal, Blogger, Contentful (via tools)
Number of plugins[61]	>=177	>=141	>=24

[56] Graphical admin interface (jekyll-admin plugin) (https://github.com/jekyll/jekyll-admin) is in development and scheduled to be released after 23th August 2016
[57] WordPress2Jekyll plugin exists which allows to export from WP to Jekyll
[58] Smart hosting services like GitHub Pages and Netlify are able to deploy to production directly from git
[59] https://gohugo.io/tutorials/automated-deployments/
[60] https://import.jekyllrb.com/docs/home/
[61] "Tools" in case of Hugo

Number of themes	>=236 [62]	>=65 [63]	>=97 [64]

Programming language. Jekyll is built on Ruby, which is well known to many developers – and it quite easy to write plugins for missing features. At the same time, the solution lacks performance and relies on the ecosystem dependencies. Hexo is built on Node.js, which is good news for JavaScript developers. However, Node is a remarkable dependency. Hugo is written in static, compiled Go language and distributed as a binary. It has no plugin system and should be plugged into developer's own build flow to add an asset pipeline. Also there are not that many Go-experienced developers which means Hugo is not that easy to extend. At the same time Hugo offers superior performance compared to other solutions.

Templating languages. Liquid and Go templates do not have that many dynamic features and thus adding dynamics should be done via extending. In this sense, Hexo offers most flexibility out-of-the-box with EJS.

Content markups. Markdown is the standard and all support it. Jekyll and Hexo support many more, while Hugo offers a limited choice.

Admin or editing interface with a GUI. Jekyll-admin is in the works for Jekyll. Hexo is oriented for blogging, and blog post creation/editing is possible via plugins. Although it is oriented for a single blog and local development environment as from GUI there is no easy way to launch to production. Hugopit offers more powerful CMS-like functionality for Hugo, but it is still a local environment tool, not for production deployments. Graphical user interface for content editors on an admin server looks to be possible, but needs a custom setup.

Preview mechanism. All provide their local server and watch filesystem for changes, offering auto-regeneration. Live browser reload (after regeneration seeing changes in browser without having to manually refresh the page) is native in Hugo, others have it through plugins. In the sense of a quick turnaround time the smoothest is to work with Hugo as site regeneration is fast.

Custom content types, data-driven content. Hexo offers a clearly blog oriented post-based approach and no data-driven content (i.e. displaying data from a YML, CSV or JSON file). Jekyll and Hugo are more general and flexible, especially Hugo as it offers a completely flexible layout and sections as well.

Custom sections, custom includes. Here Hugo and Jekyll offer flexibility, not limiting the sections to header-footer-content style breakdown. If going to details, Hugo is built from ground with a flexible layout approach and has the smoothest mechanism.

i18n support. Localization and multi-language support (translations support) of the generated site exists in all of the systems, but Hugo's system is the most advanced and has tutorials to support it.

Built-in asset pipeline. Here in basic case Jekyll and Hexo support running preprocessors etc. natively or via plugins and thus could be built without external build tools. Hugo offers just plain static output out-of-the-box, meaning developers must create Grunt, Gulp or

[62] http://jekyllthemes.org/
[63] https://hexo.io/themes
[64] https://github.com/spf13/hugoThemes

other build scripts around Hugo if they want to use preprocessors, compilers, minifiers etc. Large sites would use custom scripts anyway, so it should not be considered a huge issue – though for small sites a built-in asset pipeline would be convenient.

Atomic regeneration (incremental regeneration – regeneration of changed parts of the site only, not full site each time, to reduce turnaround time) and **regeneration speed**. Jekyll has an experimental support suitable for development environment. Hexo and Hugo do not have such capability. Hexo offers some alternative ways to speed up rebuilding, like caching [29]. One can argue Hugo does not desperately need atomic regeneration as it is already order(s) of magnitude faster than the others [30].

Deployment support. Production deployment is meant to be done via CLI (command line interface). If the development flow is from local to production then it would be convenient to configure production target and launch. For this purpose Jekyll and Hexo support many deployment targets (hosting providers). Hugo has only SFTP and AWS Lambda targets (naturally meant to build locally). However, more advanced approach would be to deploy to production via webhooks watching Git repository (configurable locally or in Git hosting platforms, e.g. GitHub, GitLab, BitBucket). This approach can be used independently of the SSG as it sits on Git repository and works with any solution.

Migration tools. The possibility to migrate a WordPress or Drupal site to a SSG is important for easy adoption of SSGs. Jekyll supports tens of different imports via plugins, and the rest support also some (all including at least WordPress). One of the ideas behind Jekyll is that it lets any normal static website be a valid Jekyll project. A site could be converted to a Jekyll with the following steps [20]:

1. Take a plain HTML mockup of a blog.
2. Get rid of repeating headers, menus, footers and so on by working with layouts and includes.
3. Turn pages and blog posts into Markdown, and pull the content into the templates.

Number of plugins. The plugin ecosystem is native for Jekyll and Hexo, as Hugo supports independent external tools, not plugins – thus has less to choose from. The quality of plugins however would need to be analysed separately and likely many are already out of date. Plugin sorting/filtering is not that easy on SSG websites and plugins get easily outdated as with WordPress.

Number of themes. Theme ecosystem is important for general adoption of SSGs. For starters, there seems to be enough to choose from.

Summary

As a conclusion, all these SSGs have their specialties – Jekyll for its most evolved ecosystem, Hexo for its blogging capabilities and Hugo for its performance and universal approach. At the same time, all are focused to what a SSG does – taking plain-text based templates, content and configuration and building these to a static site. It means that to support a full process (and other types of people in addition to developers) from development to content editing to production deployment SSG have to be combined with other tools in a custom way – and even then, convenient GUIs for content admins are still far off.

Out-of-the-box SSGs are suitable for technically-savvy site owners (mainly, modern bloggers) and have convenient support for maintaining a single website.

Some SSGs with alternative content editing and publishing methods

There are platforms that provide a web interface for creating, editing and deleting files directly on a GitHub repository, offering a WYSIWYG editor for Markdown to create a friendly composition interface. Examples are Prose, a free and open source solution, or the more advanced CloudCannon, a commercial product that allows users to edit entire sections of a static site and see a live preview of the changes. Lektor is a new SSG with a GUI editor.

There are also mobile apps, available for both iOS and Android. These are a viable option for people interested in writing and publishing content on the move. The apps connect with GitHub and the changes are instantly pushed to the repository.

Another option is to set up a service that allows users to post to a static blog by email, which can be a viable solution for those that need to constantly write on the move. It works by listening for emails on a certain address and picking up the post metadata from the subject line, the images from the attachments and the post body from the message itself [9].

3.2 Combining existing static site tools to support full process

Tools such as Contentful, Prismic.io, GatherContent and Webhook.com decouple the CMS layer from the actual website builder. This makes them suitable tools for multi-channel content management, where you're writing content not just for a particular website, but also for a mobile app, a Facebook page or a white paper. Publishing new content triggers a webhook in a build system; then, a static website generator runs the build and fetches data from the content API; and the result is pushed straight to a CDN [21].

Smart static hosting platform Netlify with its fast CDN cache invalidation for HTML and features like 301 redirects, headers, basic authentication etc. has lots of promising features for hosting a website. It can also trigger build and deploy if a change in Git repository happens and has an admin interface for hosting (but not yet for content – though experimental Netlify CMS can be tested on their website). Combining Netlify via a custom process and build configuration with a SSG and/or possibly a graphical interface like Contentful could make single site administration convenient.

Adding dynamic features. Commenting can be done via external APIs and JS widgets [like Disqus]. For form submission no plugins were found for the listed SSGs. Manually calling, for instance, a slack webhook as form action is possible and paid options like FormKeep or Netlify (with Netlify paid hosting) are possible.

Multi-site can be achieved with some configuration for instance in Hugo[65].

An example

A good example how static tools have been combined to support full process is Carrot.is combining Contentful with Netlify[66]. They achieved a process where content editing is done via Contenful static CMS (with a GUI) and site core is being built by developers on Roots static site generator. Both keep their files in Git repositories and change in master triggers Netlify which automatically triggers build and subsequently deploy to CDN.

[65] https://discuss.gohugo.io/t/hugo-multisite-workflow/103
[66] https://carrot.is/coding/static_cms.html

4. New trends and ideas for further development of static website tools

In this chapter will be discussed:

1. New trends (known fresh, beta or in-development tools)
2. Author's own ideas

4.1 Upcoming and trending tools

As mentioned in the previous chapter graphical CMSs that generate static output are coming up. A list of tools to look at are: Webhook, Contentful, Prismic.io, Netlify CMS (experimental), Lektor, jekyll-admin.

For reusability, templating and convenient publishing of new content based on existing content templates/custom types – for instance charts, quizzes, image galleries etc. one could use Autotune[67]. Autotune sits on top of a SSG generator, but adds a GUI with reusability and publishing logic.

Trends indicate that combining different tools for a full process with static sites is pretty much possible now and looks even better soon with different new tools coming up. But the setups for combining these are still quite custom to build and require some development. The problem with that is – developing and maintaining custom solutions takes time and is costly. Thus a ready set of tools to support full process would be desirable.

4.2 Author's ideas of an "ideal" platform

What developers like about SSGs and all of the modern static toolset is that they are developer-friendly, have APIs, do not need constant security patching etc. But for non-developers the appeal is not obvious yet. Perfect would be to combine these two and take an approach of a "developer-friendly CMS". It would be simple but powerful with covering most basic use cases and at the same time benefit from static's potential low-cost. So ideally, it could offer, compared to current dynamic solutions, a better quality hosting with a lower cost for all the millions of microsites out there.

In order to target the current benefits of dynamic CMSs and move significantly more of the web to static some mostly unsolved issues need to be targeted:

- Proper web-based GUI for full content (and preferably site structure and page metadata) administration is a must, including publishing content changes to production without having to use a CLI tool (editor should support different content types incl. WYSIWYG without having to write Markdown)
- A hosted simple all-in-one blogging and landing page solution is needed to appeal the hundreds millions of blog or personal site owners
- Native or plugin support (and at least configuring of existing components from GUI) for basic user-input functionalities (e.g., contact/registration forms, search, comments)
- Multi-site administration (at least dashboard for quick access – needed for freelancers building a large number of small pages, web development agencies and multiple-site owners)

[67] http://product.voxmedia.com/2015/7/8/8907841/introducing-autotune

- Support for a web-based staging environment (to conveniently support more than one content editor and without having to set up a local environment for content editing), timed publishing, users and roles, content versions comparison and rollback etc.
- Native support from existing hosting providers

Additional ideas include:

- Support for i18n of the site (localization and translations from GUI, with spreadsheet export-import capabilities in order to support translation companies)
- Support for i18n of the admin portal itself (for instance, WordPress admin is available in tens of languages)
- Support for editable div (on-site content editing)
- Full-text search in admin portal (to quickly find the component that needs to be changed)
- Support for automatic generation and handling of responsive images
- Cloning and copies of sites, templates, content directly from admin GUI, kickstarting new projects
- 1-click theme change (for instance, on WordPress.com, themes are changed several million times a year [14])
- Image gallery via a native plugin for many sites publishing image content
- A simple online store
- Data-driven content type to support partial / lazy loading, infinite scroll
- Plugin support with an advanced UI to search and filter plugins
- Possibly domain registration and nameservers (can be a built-in tool via external APIs)

Of course the hosting and content admin interface should include most of what is included in Netlify and Webhook, and supported by SSGs.

How to reach this ideal? As Hugo is the most universal and fastest among popular SSGs (suitable for developing different types of websites and offering a faster write time than others), possibly a richer toolset could be built around it. Maybe a hosted solution for both content administration and hosting (incl. an interface with a GUI) could be developed – to be easily adopted by a general audience and offer good support for content editors. At least out-of-the-box support for basic dynamic functionalities should be added as well.

Webhook looks like is a simple and powerful drag & drop CMS tool for basic use cases and webhook.com offers a hosted version of it. As it is open source, it could be forked to add new stuff – as, for instance, it does not natively support any dynamic/user-input components.

Thoughts about hosting. Even if Netlify is a smart solution, it is also more oriented for "premium" websites. For micro-site developers who need a password-protected test/staging site, the plan including password protection is starting from $9/month[68] – which is about double the price a simple WordPress hosting would cost. Netlify's free plan can be appealing to blog and small/personal site owners by its cost – but what is then lacking is that general small and blog site owners are either relying on WordPress developers or create and manage their site themselves which both require a significant learning curve with Netlify. Yet – as Netlify's free plan includes the possibility to use a custom domain (i.e., your own domain like something.com, not something.netlify.com), the free plan is actually usable. Webhook.com is not that brilliant on hosting capabilities, its free plan

[68] https://www.netlify.com/pricing (11th Aug 2016)

does not include a custom domain and it works based on zip files, not direct Git connection. For $9/month, a custom domain is possible.

Deployment model to support full process

What would be the work process with the "ideal" tool? Figure 3 proposes a deployment model with user roles, code and content flows.

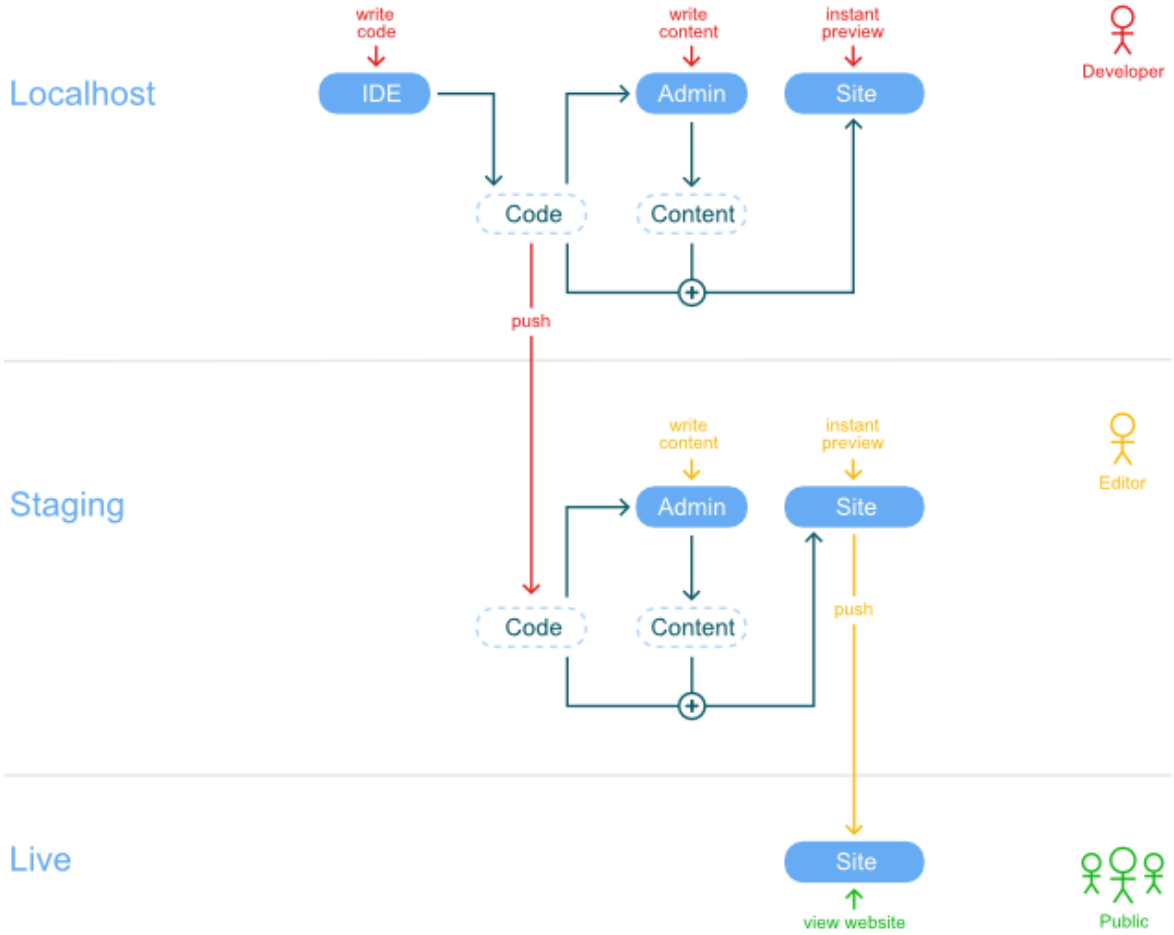

Figure 3. A possible deployment diagram to support full website lifecycle with development, staging and live (CDN) environments.

This kind of model supports flow from development to staging to live – with the possibility for content editors to work only in staging admin environment. Imagine this as a web-based multi-user multi-site admin portal. Content and site code would, of course, be kept in Git repositories. The solution would allow developers to technically manage multiple sites and content admins to manage content of multiple sites.

5. Conclusion

The goals of this study were to give an overview of the current tools available for developing, managing and hosting websites, to analyse the differences between dynamic and static websites – and to dig deeper into static website development tools.

Website types – static and dynamic – were introduced and their differences explained. For a memorable summary, static websites are like compiled programming languages – they will need time for compilation but will run fast afterwards. Dynamic sites are like interpreted languages – no compilation is needed and some flexibility is added, but performance-related and other problems might occur runtime.

Current trends of the main involved technologies and service providers were overviewed – HTTP servers, clients (browsers), static solutions, dynamic solutions, hosting providers and platforms and CDN providers. Examples which technologies are used on some websites were given.

With HTTP servers, it was found that even as Apache Web Server is the most popular, the growing trend is to use Nginx. An important benefit of Nginx is its faster static asset serving capability.

With clients and browsers, it was found that mobile devices are now as popular means of browsing websites as PCs. So no future public website can get by without proper responsive support, working well on both mobile and desktop.

Looking at static solutions and static hosting platforms, the emergence of SSGs like Jekyll, Hexo and Hugo, and smart static hosting services like GitHub Pages and Netlify was pointed out.

Dynamic websites are mostly built on PHP and WordPress is the dominant platform running 26% of world's active websites. Pretty much all of the world's popular "classic" hosting providers support WordPress out-of-the-box.

CDN providers like CloudFlare and Akamai serve the assets of the world's busiest websites. Smart static hosting services GitHub Pages, Netlify, Aerobatic use a global CDN by default (as opposed to "classic" hosting providers which usually do not). Using a CDN for any relevant website is recommended by the author.

Analysing the benefits of static and dynamic website technologies, we described the core advantages of static websites – performance, efficiency and security.

We found that static tools are modern, but developer-centric. Easy-to-use graphical user interfaces for website content editors, bloggers etc. are still limited and one would need custom setups to combine different tools.

The most popular Static Site Generators (SSGs) Jekyll, Hexo and Hugo were compared. They all do quite the same thing – generate static websites as files and directories. Which is just one of the steps in a website's lifecycle. As the main difference, Hexo is oriented on blogging and does it well – while Jekyll and Hugo have the potential for generating all kinds of websites. Jekyll is written in Ruby and has a usable plugin system. Hugo is written in Go and can only support external helpers. At the same time, Hugo has superior performance which would is suitable for larger websites and is a pleasure to work with.

SSGs are a part of an emerging static websites ecosystem which is looking to be expanding and growing into wow-effect generating toolsets. Some ways to combine existing tools

are pointed out. Some tools which are already meant for more than working with websites locally are mentioned – and some ideas where to evolve with the "ideal" tool are stated.

The leading dynamic CMS WordPress has had many interesting developments in the past couple of years as well. There are plugins that support converting WordPress site to static keeping the original admin interface (e.g. Static HTML Output Plugin and WordPress2Jekyll plugin). WordPress.com has created a new admin interface Calypso which is implemented in JavaScript frameworks and runs mainly on client-side, connecting with the PHP backend via a REST API[69]. Calypso is usable from build tools via an API like modern static tools.

However, even if WordPress is advancing, for most use cases the author suggests using a static approach and adding dynamic features as needed, not vice versa – making a dynamic site and trying to cache it with heavy machinery. Static site can save time and money, as it requires less maintenance and less server resources. Static sites are fast, scalable, can handle high volumes of traffic and are more secure against hacking.

The author's view is that the future of most websites will be static. The benefits of static are clear and as the toolset is maturing the shortcomings will be solved. Ongoing static shift offers possibilities for platform developers to develop more complete and hosted static platforms – and for "dynamic" plugin/services developers to enable support for dynamic functions in static websites.

Wider adoption of static technologies for static-by-nature websites would simply make the Internet a better place – reduce the possibilities of cybercrimes due to less compromised servers, provide better user experience and generate less negative impact on the environment.

5.1 Future work

Static tools are in constant development and new parts of the toolset could be reviewed in more detail. Webhook, Prismic.io, Contentful and others provide graphical user interfaces. Jekyll-admin and Netlify CMS are in development. Out-of-the-box static toolsets combining different services could be created.

The workflow and technologies used by popular SSGs differ somewhat from what PHP developers (e.g. WordPress developers) are used to. An experienced WordPress developer who has migrated to a static solution could think about what can be done to support wider migration.

More detailed experiments and analysis could be done about the performance and scalability of static vs dynamic sites. Total Cost of Ownership (TCO) analysis could be done about developing, hosting/administrating and scaling static vs dynamic solutions, with taking security risks into account.

As dynamic CMSs are also under development and are likely to maintain the advantages for a few use cases, a comparison of the new capabilities and analysis of recommended use cases for static and dynamic could be done in some time from now.

[69] http://ma.tt/2015/11/dance-to-calypso/

6. References

[1] Wikimedia Foundation. Website. [Online]. https://en.wikipedia.org/wiki/Website

[2] Verisign, Inc. (2016, July) Internet Grows to 326.4 Million Domain Names in the First Quarter of 2016. [Online]. https://investor.verisign.com/releasedetail.cfm?ReleaseID=980215

[3] Netcraft Ltd. (2016, July) July 2016 Web Server Survey. [Online]. http://news.netcraft.com/archives/2016/07/19/july-2016-web-server-survey.html

[4] Wikimedia Foundation. World Wide Web. [Online]. https://en.wikipedia.org/wiki/World_Wide_Web

[5] Wikimedia Foundation. Static web page. [Online]. https://en.wikipedia.org/wiki/Static_web_page

[6] Wikimedia Foundation. Dynamic web page. [Online]. https://en.wikipedia.org/wiki/Dynamic_web_page

[7] WordPress.org. WordPress Plugins. [Online]. https://wordpress.org/plugins/

[8] Wikimedia Foundation. Content delivery network. [Online]. https://en.wikipedia.org/wiki/Content_delivery_network

[9] Eduardo Bouças. (2015, May) An Introduction to Static Site Generators. [Online]. https://davidwalsh.name/introduction-static-site-generators

[10] Justin Ellingwood. (2015, January) Apache vs Nginx: Practical Considerations. [Online]. https://www.digitalocean.com/community/tutorials/apache-vs-nginx-practical-considerations

[11] StatCounter. (2016, August) Comparison from July 2015 to July 2016. [Online]. http://gs.statcounter.com/#all-comparison-ww-monthly-201507-201607

[12] W3Techs. (2016, August) Usage Statistics and Market Share of Content Management Systems for Websites, August 2016. [Online]. https://w3techs.com/technologies/overview/content_management/all

[13] Karol K. (2016, March) Writing About WordPress? Here's Where to Get Your Statistics From. [Online]. http://www.codeinwp.com/blog/wordpress-statistics/

[14] Michelle W. (2015, January) Im(Press)ive! Your Year in Review. [Online]. https://en.blog.wordpress.com/2015/01/06/2014-in-review/

[15] W3Techs. Usage of reverse proxy services for websites. [Online]. https://w3techs.com/technologies/overview/proxy/all

[16] Chris Bach. (2015, September) Instant Cache Invalidation. [Online]. https://www.netlify.com/blog/2015/09/11/instant-cache-invalidation

[17] Tony Mauro. (2015, March) Why Netflix Chose NGINX as the Heart of Its CDN. [Online]. https://www.nginx.com/blog/why-netflix-chose-nginx-as-the-heart-of-its-cdn/

[18] Michael C. (2016, January) Nginx Vs Varnish Vs Apache Traffic Server – High Level Comparison. [Online]. https://www.bizety.com/2016/01/07/nginx-vs-varnish-vs-apache-traffic-server-high-level-comparison/

[19] WordPress.com. Posting Activity. [Online]. https://wordpress.com/activity/posting/

[20] Mathias Biilmann Christensen. (2015, November) Why Static Website Generators Are The Next Big Thing. [Online]. https://www.smashingmagazine.com/2015/11/modern-static-website-generators-next-

big-thing

[21] Mathias Biilmann Christensen. (2015, November) Static Website Generators Reviewed: Jekyll, Middleman, Roots, Hugo. [Online]. https://www.smashingmagazine.com/2015/11/static-website-generators-jekyll-middleman-roots-hugo-review/

[22] moz.com. Page Speed. [Online]. https://moz.com/learn/seo/page-speed

[23] Sean Work. (2011) How Loading Time Affects Your Bottom Line. [Online]. https://blog.kissmetrics.com/loading-time/

[24] Alan Shimel. (2013, June) 7 of 10 leading WordPress plugins are vulnerable. [Online]. http://www.networkworld.com/article/2224843/opensource-subnet/7-of-10-leading-wordpress-plugins-are-vulnerable.html

[25] Teemu Koskinen, Petri Ihantola, and Ville Karavirta, "Quality Of WordPress Plug-Ins: An Overview of Security and User Ratings ," in *ASE/IEEE International Conference on Social Computing and 2012 ASE/IEEE International Conference on Privacy, Security, Risk and Trust*, 2012, pp. 834-837.

[26] Matthew Prince. (2014, February) Technical Details Behind a 400Gbps NTP Amplification DDoS Attack. [Online]. https://blog.cloudflare.com/technical-details-behind-a-400gbps-ntp-amplification-ddos-attack/

[27] Albert Ahdoot. (2014, September) Are The United States' Data Centers Wasting Energy? [Online]. http://www.colocationamerica.com/blog/energy-wasting-data-centers

[28] Chris Bach. (2015, September) Continuous Deployment. [Online]. https://www.netlify.com/blog/2015/09/17/continuous-deployment

[29] Hexo. (2016, February) Hexo 3.2 Released. [Online]. https://hexo.io/news/2016/02/28/hexo-3-2-released/

[30] Ludovic Chabant. (2015, July) Multi-core PieCrust 2. [Online]. Multi-core PieCrust 2

Non-exclusive licence to reproduce thesis and make thesis public

I, Hillar Petersen,

herewith grant the University of Tartu a free permit (non-exclusive licence) to:

1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

From Static and Dynamic websites to Static Site Generators,

supervised by Karl Blum and Tõnu Tamme.

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, 12/08/2016