Introduction to the Static Site Generator (SSG)

Starts with the design goals and a review of the rationale for a web site content manager. It then covers installation and the instructions to adapt the program to serve your own homepage!

August 4, 2023

Contents

The programs to generate a web site[1] available in 2018 did not satisfy my expectations but I felt that most of the tools required to build a static site generator where available. Thus I embarked on building **Yet Another Static Site Generator** adaptet to the needs of an academic[2].

- searchable list of papers ready for download,
- texts readable in a browser but printable as `pdf`.

After a short introduction to the Static Site Generator SSG follow the instruction to download and to adapt the program to produce a personalized homepage. Not much to do other than organizing content pages in directories, include title and abstract to each content page and add a title and abstract tot the `index.md` page in each directory.

[1] a.k.a. content management systems

[2] The web contains a surpsing amount of advice,from - a consultant, or older, from 2012, by publisher, or - current to go to a static site generator and markdown, 2, - yet another commerical service and - another howto

The Static Site Generator

My goals for daino. What is different from other Static Site Generators? The design goals and rationale and functionality testing.

My Goals with SSG

My Goals with SSG

The use case is my own homepage[3] with requirements typical for an academic researcher. and should be built from available packages in Haskell[5].

- Use an (inexpensive) host server[6].
- Allow me to use a page layout following Tufte css.
- Force a strict separation of *content* and *presentation*[7].
- Look for simple handling and long term stability.

[3] I will use the term **site** (or *web site*) for a set of connected **web pages** (or just *pages*) which can be accessed through a web browser using the `world wide web` technology[4].

[5] especially `pandoc` and `shake`, which reduces the effort to maintain the code using using my "uniform" approach to wrap packages in integratable interfaces.

[6] There are some servers free of charge, e.g. github or google, but I prefer indepence and looked for a basic web server, which cost me Euro 3 per month.

[7] here called `dough` and `theme`, which is baked into the web site.

Installation and test for functionality

Installation and test for functionality

I have tried a number of web site generator programs and found that a test installation and checking the methods for customization is the fastest way to identify what suits my requirements.

Such a test consists of two steps: (1) install and check funtionality of installation with a test[8] and (2) build your site.

[8] The `hello world` test for a site generator.

Installation and basic test for functionality

A basic test for functionality is:

- Clone or copy the code from github[9].
- Change into the `ssg` directory and install with `cabal install` (or perhaps better with stack `stack install`) which produces `ssgbake`, the program which converts (*bakes*) the content into a static site.
- Run `ssgbake -tsw` which produces a test site in your home directory `~/bakedhomepage` which is served on port 3000.

[9] `git clone https://github.com/andrewufrank/SSG`

- Open in your browser `localhost:3000` and you should be greeted by the landing page of the test homepage.
- Edit, for example, the file `ssg/docs/site/dough/Blog/01blog1.md` and observe how the web page is adapting (after refreshing the browser cache!).

If you are satisfied that the installation works, you can proceed to build your own site!

Build your site with SSG

The steps necessary to build a site.

Build your own site

Copy the content of the `ssg/docs/site` directory to where you would like to locate your homepage and rename it to `myhomepage` or whatever directory name you fancy.

I would start `git` in this directory to achieve a flexible backup on the site with `git init`. A suitable `.gitignore` is already in the copied directory and may required adaptation.

Adapt the file `ssg/docs/site/settings3.yaml` minimally[10] with a editor for program text files (i.e. not `office`) for:

- the location of folders, at least for
  - dough: the folder with the source of your site
  - baked: the foler where you expect the generated site (could be, for example, `/var/web/` or `~/bakedhomepage`)
- the port the server is using, when run `ssgbake -s` (default is `3001`)[11]
- `menuitems`: the first levell of subdirectories for the web page files.

After adaptation restart with `ssgbake` in the directory of your homepage and the homepage will be produced, adapted to your needs.

Customization is

- in the `settings` file, and in
- web page files in the subdirectories to the `dough` directory.

The example site in `ssg/docs/site/dough` contains examples for the `settings` file and for web pages with solutions for different uses, e.g. references to images, literature.

All easily customizable aspects are in files and no new compilation of `ssg` is needed[12].

Under the `dough` directory you can include content, typically organized in subdirectories. Each web page corresponds to one file, including the files linking other files in subdirectories.

[10] For more details

[11] The possible switches are - `-s` to start a server, - `-q` for quick, meaning not to produce pdf files, - `-w` to watch files changing and re-bake them automatically.

[12] Recompilation may be needed for new versions of `ssg` or new versions of compilers; it is recommended, but probably not required, to delete the baked website and rebuild it completely.

The overall setup of a site

The file describing the overall setup of a site.

Details of the Settings for a site

Details of the Settings for a site

I will use the term **site** (or *web site*) for a set of connected **web pages** (or just *pages*) which can be accessed through a web browser using the `world wide web` technology[13].

The settings are all collected in a single YAML file[14]. The annotated file for the currently running site can probably serve as a concrete example.

The settings start with `siteLayout`, which gives the directories of the sources for

- theme: where the details of the appearances of the content are fixed,
- dough: the source text for the web pages,
- baked: where the converted files for the web site go; this may be `/var/www/html`[15],
- masterTemplateFile: the template which determines the layout of the converted html - probably use the one provided and adapt later if necessary.
- blogAuthorToSupress: name or names of the authors of most of the material on a site, which should not be repeatedly shown as authors

The content must use the keywords that the theme set up; it is possible to produce with the same theme (i.e. the same directory with the same files) different web sites from different source directories. It is likewise possible to produce different `baked directories` which are independently served from different theme and the same content files.

The `localhostPort` gives the port used by the server created with the `-s` switch of `ssgbake`.

The `siteHeader:` needs values for `sitename:`, `byline:`, `banner` (an image[16] to place by default at the top of all pages) with a `bannerCaption`, a text which can be read if the image not visible.

Last, the entries of a *static menu* are given as `menuitems:` which is shown as a ribbon under the banner page. They consist of a

[13] Following the seminal ideas of Tim Berners-Lee

[14] The current specification of YAML, but there are perhaps better explanations

[15] The default web root for NGINX

[16] preferably wide and narrow; 1024 by 330 pixels works well

- `navlink`: wich is a relative adress to a directory, usually within the `dough` folder.
- `navtext`: the text shown for the link.

The settings file is read each time `ssgbake` is started and content is `baked`; changes are burnt into the converted site and after changes, the site should be rebuild[17].

[17] Just delete the `bakedHomepage` directory and rebuild with `ssgbake`.

Topical subdivision of content

Topical subdivision of content

Usually the content of a site is divided in some topics, e.g. `contact`, `publications`, `blog`. The content for each topic, i.e. the markdown files, are collected in these directories.

Additionally an `index.md` file must be added, which serves as a introduction to the content; a sort of `table of content` is appended automatically and facilitates navigation with clickable links.

Landing page

Landing page

The landing page, i.e. the page shown when the URL of the site is opened. It typically contains a general introduction and links to the major pieces - possibly with some explanation.

The *landing page* of the homepage will be produced from the file `index.md` in the root (`dough`) folder of your homepage using the theme given in the settings file; no special rules or provisions!

Resources directories

Resources directories

Directories to include resources[18], e.g. images or pdf files[19], which are references in other web pages and served can be added wherever convenient. Their location are mentioned in the references included in the source texts for the web pages they reference.

[18] `resources` is a reserved name for directories in SSG; these directories are not searched for web content and should only contain static content, which is references from other pages.

[19] currently only files with extensions `jpg`, `JPG` or `PDF` are dealt with, but extension is a simple change in the Haskell source, specifically in `Shake2.hs`.

The structure of the page files

The structure of the source files for the web pages consist of a header (using YAML syntax) and the page content written in markdown.

The `YAML` header

The `YAML` header

The first part of each web page describes the page. It is fenced off from the page content proper by `---` lines above and beyond. It follows the `YAML` syntax:

```
---
title: text which becomes the title of the page
abstract: typically a multi line text describing the page.
        It becomes the abstract of the page and is shown
        together with the title on the index pages.
author: the author of the page,
        there is a mechanism to suppress this
        for the author of a site
        ([see](/Essays/SSGdesign/004settings.html))
keywords: some descriptive keywords.
date: 2019-03-05
image: if present a reference to the image file
        which will become the pages banner
        (if blank, the default site banner image is used).
bibliography: a reference to the `bib` file
version: publish or draft
visibility: public or private
---
```

Web page content

Web page content

It is followed by the text written as markdown.

- titles are marked with `#` and `##`, which give second and third level titles[20].

  For more details of the (Pandoc) markdown syntax see.

[20] The text after the `title:` keyword in the header gives the first level.

Index pages

Index pages

The structure of the site is revealed to the user through `index pages`[21]. They list the titles and abstracts of the web pages included in a directory, starting from the `root` in a hierarchy. The pages are `clickable` and permit navigation[22].

The index pages must be started by the author of the site as a file `index.md` with keywords

- indexPage: true
- indexSort: title

where the `indexSort` field indicates the order in which pages are listed. A sort by `title` sorts the pages by their filename, which permits to use filenames starting with a number to achieve a specific order.

Alternatives are sort by `data` or `reverseDate` (newest first).

[21] `index.html` files

[22] In addition to the ribbon under the banner image which is always linking to the major subdivisions, listed in the `settings` file and clickable `sitename`.

Referencing images and other static content

Referencing images and other static content

The references can be either absolute to the web root[23], i.e. the directory in which the `dough` is placed or relative to the location to the current page file[24].

Remember that the references must include the `.html` extension of the files in the baked form and not the `md` extensio of the original content files.

It is often useful to place the static content in a `resources` directory[25] in the same directory as the pages for a topic.

[23] I.e. starting with "".

[24] The directory name, not starting with "".

[25] with exactly this name!

Pages rendered as PDF

Pages rendered as PDF

For every web page transformed to `html` a corresponding `pdf` is produced, using the `KOMA` tools for `latex` and rendered as a `scartcl`.

The pdf format uses footnotes at the foot of the page, whereas the footnotes in the web output are pushed to the margin[26]. The bibliography in both output formats are at the end of the page.

[26] Tufte style

Part I

Internal documentation

The internals of the Static Site Generator (SSG).

Some of the more interesting aspects of the internal design of `daino` are given here; mostly useful when improving the code.

Produced with 'daino' (Version versionBranch = [0,1,5,3,3], versionTags = []) from /home/frank/Desktop/myHomepage/Essays/SSGdesign/03introUse/index.md with latexTufte81.dtpl arguments booklet