Website Documentation | Mobile Bay DSA

Last Edited 10/16/2025

Contents

Part I

Hosting

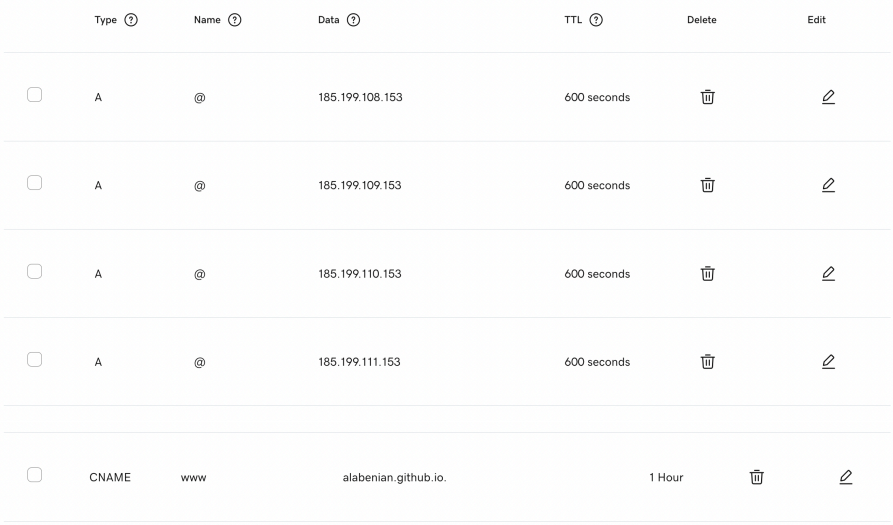
Introduction

Our website is built with Jekyll, Tailwind, and Markwhen; hosted with Github Pages from https://github.com/alabenian/mobilebaydsa; and presented to https://www.mobilebaydsa.org/ for the public to see. This little book exists to help members work on the site and consider any alternatives to the current setup if needed.

Chapter 1

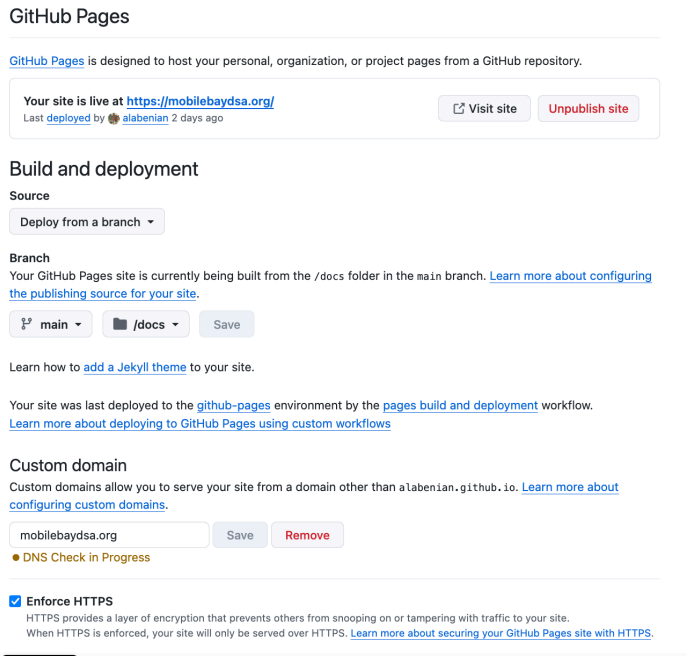
Basic Hosting Information

Our domain name `mobilebaydsa.org` is registered through Godaddy and currently presents the Github Pages site at https://alabenian.github.io/mobilebaydsa. It could present any other source, but I chose to use Pages because our Godaddy plan did not come with a site builder and other options would cost some money while Pages was free, and because it gives us complete control over the site as long as it's static (i.e. as long as we don't need to store data on the fly, and all processing can be left to the user's computer, unless an additional outside service is used) and its contents are safe to keep public. Godaddy is told where to get the website using four `A`-type "@" DNS records which give it the IP range of Github and one `CNAME`-type "www" record which tells it to use `alabenian.github.io`:



All this is available from Domain/DNS/DNS Records at the `mobilebaydsa.org` control page in Godaddy; a more in-depth tutorial is available at this Medium page. Part of the original process involved also deleting the DNS records for the fallback site generated by Godaddy.

Github makes things easy to set up on its end: we publish the site to Github as a repository (https://github.com/alabenian/mobilebaydsa), enable Github Pages, which will automatically deploy it to `[username].github.io/[repository name]`, then tell it to use the domain name `mobilebaydsa.org` and enforce HTTPS:



This should then automatically add a `CNAME` document containing "`mobilebaydsa.org`" to the repository. The only other setting I've changed here is to have it deploy from the `/docs/` folder of the repository rather than the root. Here `/docs/` is just a renamed version of the `/_site/` output of Jekyll; this is deployed as a static site pre-built by Jekyll locally rather than one where Github runs Jekyll itself after changes are made to the repository, due to an incompatibility with one of the plugins I was using for our events posts (jekyll-leaflet). *I will explain this more later, but for now, it's not relevant.*

1.1 Github Pages and Potential Alternatives

Github Pages is a nice, free choice of hosting which gives us maximal control over the website as mentioned before, but it can be somewhat daunting and even inaccessible to anyone who isn't already used to working with websites

this way. The biggest con besides this is the fact that, on the free plan, all contents of the site have to be public unless an external service is used to store private, member-only information. Despite the extra difficulty of getting set up for those who would already have access, it's arguably more democratic than any other hosting service – changes are published to the site when someone makes changes on their own download of the repository and sends them to Github through a **commit** with details of the changes they made to a local **branch** via the Github Desktop app (or via the command line), before making a **push request** that, when accepted, will publish them to the **main** branch and thereby the official website. The push request can be accepted by the owner (currently "alabenian," but could be transferred to an official Mobile Bay DSA account) or anyone who has been listed as a collaborator on the project. This setup has the benefit of allowing *anyone* to suggest changes to the website to be confirmed by authorized accounts. This way everyone has a voice to publish blog posts, new pages, design ideas, etc. to the site such that official collaborators can see every change and prevent security problems. I cited inaccessibility of editing as the main downside, but that's not too hard to get around either: with some effort I can make a separate service that allows for limited visual editing and makes automatic push requests for those changes.

One alternative would be a site builder like Wordpress.com which allows easier visual editing and supports blogs, etc. the same as Jekyll. Wordpress is especially nice because it would allow us to include member-only content on the website without a separate service, secured either by password or a whitelist of Wordpress accounts. Wordpress does not allow free sites to use custom domains, however, and the cheapest plan that *does* is priced at $96/year. Godaddy has a forwarding feature that would let us send anyone at `mobilebaydsa.org` to the Wordpress site, though I don't know how this would affect the site's presence in Google searches. The other major differences to consider are restrictions on design (which can also be removed with a $24/year custom CSS add-on) and a Wordpress watermark that comes with the free plan (I've seen this on other DSA chapter sites, and it's not *too* bad).

Part II

Basic Structure & Working on the Repository

Chapter 2

Basic Setup

2.1 Getting Started

If you want to make changes to the repository, you'll first need to establish a local branch to work on. I'm writing this guide under the assumption that you've *never* used Github before, in which case I'd recommend working through Github Desktop and Visual Studio Code. After downloading those apps, open Github Desktop, set up a Github account and follow the app's instructions to clone the repository into whatever directory you'd like using its link https://github.com/alabenian/mobilebaydsa. Then open the folder with Visual Studio Code, and install the following command-line tools in your terminal (either as a standalone "command prompt" window or from the bottom terminal tab in VSCode).

There are three command-line tools in total used to build the site: **Jekyll**, **Tailwind**, and **Markwhen**. Installing Jekyll takes the greatest priority, as it is necessary to build the site at all (if we switched to having Github build the site with Jekyll itself, installing it locally wouldn't be 100% necessary, but it's still nice to be able to view what you've done before publishing). Tailwind is second, because it creates the `.css` file used by the rest of the site for styling. Because this file has already been created, Tailwind is only necessary for *editing* it. Finally, Markwhen is only used to render and update the timeline used on the Calendar page of the site, and is only necessary to someone who is adding or removing events on the calendar.

Jekyll (https://jekyllrb.com/) simplifies work on the site by structuring the files by root directory markdown (or HTML) files containing the contents of each page, `_layout` HTML files which are referenced in each to determine how the contents are presented, and `_include` HTML files which are referenced in either the layout or content files to display some extra piece of the website (e.g. a navbar or a commonly-occurring button or icon). The resulting file structure

of the site will be discussed further in "Repository Structure." It requires Ruby (installation information here) and RubyGems (which should be included with Ruby) to install via `gem install bundler jekyll` in the terminal. After installation, the site can be built by opening the repository folder in the terminal (or `cd`-ing into it; this is already done if you've opened the entire folder in VS-Code, so you can use the terminal tab at the bottom) and running `bundle exec jekyll serve`. I've used some additional plugins on the site, so it'll throw an error on the first build and prompt you to install those; from there any work with Jekyll should be smooth sailing!

Tailwind (https://tailwindcss.com/) simplifies writing CSS for the site by supplying a set of classes with short and easy-to-remember names. It can be installed as a CLI tool with `npm install tailwindcss @tailwindcss/cli`. (This uses `npm`, so unfortunately you'll need to install Node first: https://docs.npmjs.com/downloading-and-installing-node-js-and-npm) To use it on the site, just open the folder in another terminal tab (or, again, `cd` into it) and run `npx @tailwindcss/cli -i ./styles/initstyles.css -o ./styles/finalstyles.css --watch`. It will then replace the "`@import "tailwindcss";`" in the initial styles file and watch for future changes to re-run the command in that terminal instance. This adds so much extra hassle for getting set up to edit styles, I may remove it in the future – it made the process of designing the site much easier for me because I had already installed it globally and knew its classes.

Markwhen renders the markwhen file `calendar_source/main_calendar.mw` into `calendar_source/timeline.html`, which is embedded on the Calendar page. Install it using `npm i -g @markwhen/mw` (this also requires Node – see instructions for Tailwind). After installation, all you need to do is run `mw calendar_source/main_calendar.mw calendar_source/timeline.html` to update the timeline if changes to `main_calendar.mw` have been made. See the "Events & Posts" chapter for more information about editing the timeline.

2.1.1 TLDR ("Just give me the instructions already!")

1. Create a Github account
2. Download and install Github Desktop and Visual Studio Code
3. Open Github Desktop and follow the instructions to log in and clone a repository. Use the link `https://github.com/alabenian/mobilebaydsa` or `alabenian/mobilebaydsa`.
4. Open Visual Studio Code, and within it, open the folder you created by cloning the repository in the previous step. Tell VSCode that you trust the author, then find the "terminal" tab at the bottom of the window.
5. (Windows) Download and install a Ruby release from https://rubyinstaller.org/downloads/.
6. (Mac) Use Homebrew (if you don't already have it installed, first run `/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD` to update your Mac's built-in installation of Ruby: `brew update` and

then `brew install ruby` in your VSCode terminal.

7. (Windows) Download and install a release of NVM Windows from https://github.com/coreybutler/nvm-windows/releases. Then run "`nvm install latest` in your VSCode terminal.
8. (Mac) Run the NVM install script: `curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.3/i | bash`
9. Still in your VSCode terminal, run these commands:

- `gem install bundler jekyll`
- `npm install tailwindcss @tailwindcss/cli`
- `npm install @markwhen/mw`

8. Mouse over to the "+" button to the right of the terminal and click it twice to create two more terminal tabs. In each of the tabs now open, run one of the following commands:

- `bundle exec jekyll serve` – This will prompt you to install some additional plugins. Afterwards, it will build and serve your website to some localhost port. Take note of the "server address" it gives you immediately after this, open it in your browser, and bookmark it if necessary for future reference to see the effects of your changes.
- `npx @tailwindcss/cli -i styles/initstyles.css -o styles/finalstyles.css --watch` – This will add any Tailwind classes used throughout the website to the base stylesheet, generating finalstyles.css, which is used as the main stylesheet for the whole website.
- `mw calendar_source/main_calendar.mw calendar_source/timeline.html` – This will re-render the timeline used on the Calendar page, which has not yet been edited. Adding or removing events is as simple as editing `calendar_source/main_calendar.mw` and then re-running this command, though I'd like it to be standard practice to create an event post complete with a map of the location to which each event on the timeline may link.

9. Start working!