v2.0.0 Design

- The favicon will be 1 black wire going into a hub with 3 wires leaving (1).
- The header will be larger bold text that says 'hub' (2).
- Styles will be refactored from the template into 'static/styles.css' and be as general as possible (3, 5).
- The template will be moved into a file called 'tmpl/index.html'. It will be cached using a cache like in 'Cache'. A separate process will clear the cache every day (4, 6).
- `http.FileServer` will be used to serve static files (7).
- A cache like in 'Cache' will be checked by the main handler before querying a favicon and updated after querying a favicon. A separate process will clear the cache every day (8).
- The major version will be embedded in the server. A separate process will check the Github release API for the most version in that major version every day. The release with that version will then be downloaded and extracted over the static and template directory. The Github release API will be abstracted behind an interface like in 'Release Manager' (9).

Cache

```
type Cache struct:
* Get(URL) (FaviconPath, bool)
* Put(URL, FaviconPath)
* Clear()
```

Release Manager

```
type Version struct:
* Major int
* Minor int
* Patch int
```

```
type ReleaseManager interface:
* MostRecentVersionWithMajor(Major) (Version, error)
* Copy(Version, ReleaseDirectory, ToDirectory) error
```

```
type GithubReleaseManager struct:
* ReleaseManager
```

v1.1.0 Design

- A `LoadFavicon` function will accept a `Website`, make a request to the URL, and parse an image URL at a tag with `rel="icon"` if it exists into the `Website`. This will be called for each `Website` in `Handler`. The image will be injected to the right of each website link (1).
- A shell copy config script will use `scp` to copy the config file to a passed remote host, user, and directory (2).
- A shell deploy script will use `ssh` to log into a passed remote host and user and start `hub` with `nohup` in a passed working directory (3).

v1.0.0 Design

- A `go` command with no arguments will serve a single home page with its template embedded in the binary.
  - `go get` will install the command (5).
  - The command will then be able to be immediately run with no arguments (4).
- For each request, the server opens a config file located at a fixed location, reads the directory from it, injects the directory into the embedded template, and serves the HTML page (1, 4).
  - The config file will be a YAML file with the structure in 'Config' (3).
  - The directory will be injected into a list which shows each websites name with a link to the website (2).

Config

```
- URL: <URL>
  name: <NAME>
- ...
```