v3.0.0 Design

- The URL will be parsed and the path will be stripped of before appending absolute favicon paths '/' (1).
- 'styles.css' will be updated so the page background matches the favicon background and all text matches the favicon foreground (2).
- All elements will be wrapped in a 'main' tag which will be centered using flexbox (6).
- A tested package called `cache` will be added (3, 4).
- Specific getters wrapping `cache.Getter` will be provided for each of favicons, the config, and the template.
  - A `cache.Cache` decorated with `cache.ThreadSafeDecorator`, `cache.LogDecorator`, and `cache.TimeDecorator` will be accessed by the getter for loading favicons.
  - A `cache.Cache` decorated with `cache.ThreadSafeDecorator`, `cache.LogDecorator`, and `cache.ModifiedDecorator` will be accessed by the getter for the config (3).
  - A `cache.Cache` decorated with `cache.ThreadSafeDecorator`, `cache.LogDecorator`, and `cache.ModifiedDecorator` will be accessed by the getter for the template (4).
- `Cache`s will be injected into the main `http.HandlerFunc` through a closure (3, 4).

`cache`

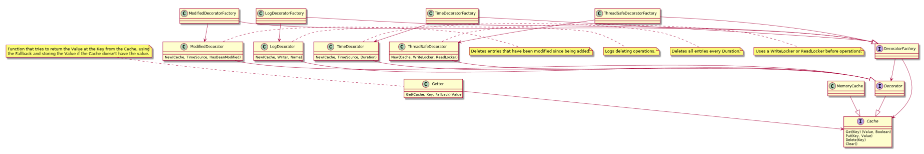
ModifiedDecoratorFactory
LogDecoratorFactory
TimeDecoratorFactory
ThreadSafeDecoratorFactory
Function that tries to return the Value at the Key from the Cache, using the Fallback and storing the Value if the Cache doesn't have the value.
ModifiedDecorator
New(Cache, TimeSource, HasBeenModified)
LogDecorator
New(Cache, Writer, Name)
TimeDecorator
New(Cache, TimeSource, Duration)
ThreadSafeDecorator
New(Cache, WriteLocker, ReadLocker)
Deletes entries that have been modified since being added.
Logs deleting operations.
Deletes all entries every Duration.
Uses a WriteLocker or ReadLocker before operations.
DecoratorFactory
Getter
Get(Cache, Key, Fallback) Value
MemoryCache
Decorator
Cache
Get(Key) (Value, Boolean)
Put(Key, Value)
Delete(Key)
Clear()

Figure 1: `cache` Design

v2.1.0 Design

- A copy assets shell script will copy 'static' and 'tmpl' to a passed remote address, user, and directory (1).

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).

Cache

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

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>
- ...
```