

COMP2772 Web-Based Systems Development

Assignment 2 - Style Guide and Specifications Document

Thomas Wade | Wade0059 | 2181554

Wednesday 31 October, 2018

0.1 Table of Contents

0.1 Table of Contents	1
1.0 Project Structure	2
2.0 Style Guidelines.....	4
2.1 Style	4
2.2 Layout.....	4
2.3 Views, Templates, and Partial	4
3.0 Functionality	6
3.1 Additional Libraries	6
3.2 Client-Server Interaction	6
3.3 Operation	6

1.0 Project Structure

The directory structure of the project is designed to keep each component of the site isolated in a clean and logical manner. Server content is stored in the root directory with data in its own subdirectory. All content intended to be accessible by the client's browser (HTML, JavaScript libraries, AngularJS, and stylesheets) is stored within the `static/` subdirectory. See Figure 1 for a more detailed directory structure breakdown.

Several routes are employed on both the client and server applications. The client application contains five routes listed in Table 1 below. Each of these routes have a corresponding controller and view.

The server has a collection of routes that provide a RESTful interface for pushing and retrieving data. These are used by the client application to retrieve site content and push new posts. Table 2 below lists each route, its HTTP method, and what it does. All data is transferred in JSON format unless otherwise noted.

```
data/
└─ <server data JSONs>
node_modules/
└─ <various packages>
package.json
package-lock.json
README.md
server.js
static/
├─ 404.html
├─ index.html
├─ partials/
│   └─ <partial HTML fragments>
├─ scripts/
│   └─ <browser scripts and libraries>
├─ styles/
│   └─ <stylesheets>
├─ templates/
│   └─ <templates>
└─ views/
    └─ <views>
```

Figure 1 - Abbreviated project directory structure

Route name	Description
/	The root route, shows the blog page containing each blog post.
/about	The about route, shows the about page containing a background of the site.
/faq	The FAQ route, shows a list of frequently asked questions and answers.
/create	The create route, used to edit and submit new posts to the site.
/*	The default route, shows a 404 page stating the requested page could not be found.

Table 1 - Client routes and explanations

Route Name	Method	Description
/*	GET	The default route, used as a catch-all to serve the index page as HTML. The client application is expected to handle 404 errors through its default route.
/api/posts	GET	The posts route, retrieves a list of all blog posts.
/api/posts	POST	The posts route, accepts a single post and prepends it to the posts list.
/api/about	GET	The about route, retrieves the content for the about view.
/api/faq	GET	The FAQ route, retrieves a list of all frequently asked questions entries.

Table 2 - Server routes, HTTP methods, and explanations

2.0 Style Guidelines

2.1 Style

The overall style of the site is designed to be clean and simple. The colour palette primarily consists of pure black and white, except for a few tones of grey used to provide emphasis on text inputs. As for fonts, Arial, Helvetica, and the browser's default sans-serif font are used. There were two goals behind this: to provide a consistent experience across various devices, and to maintain a focus on site content without looking bare.

Active elements, such as buttons and anchors, as well as headings or other notable content have their colour scheme inverted to a black background and white bold font for emphasis. The choice to globally include anchors under this was due to markdown input allowing posts to contain bold text (which was the original format for anchors), so a different, stylistically consistent, style had to be applied.

2.2 Layout

All content on the site is contained within a central column with a set width of 800 pixels. The column is centred horizontally to accommodate for large screens and will adjust on-the-fly. Should the window become too narrow the column will overflow horizontally. This guarantees a consistent experience across all devices.

Located at the top and bottom of the central column are a header and footer. The header is fixed to the top of the window and is intended to look like a pair of tabs hanging down, sitting on top of the content under it. The site title and tagline are located on the left side of the header and a set of navigation buttons are located on the right. The footer sits either on the bottom of the window or below the content, whichever is further down.

2.3 Views, Templates, and Partial

The site is broken up into several views, templates, and partial HTML files. For each 'page' that the user can interact with there is a corresponding view that determines the content within it. The header, central container, and footer are defined in the static index file and each view is placed within the central container. The header and footer are included within the index file as partials to reduce clutter and avoid unwanted side-effects when they are changed.

The contents of the central container are entirely the view's responsibility to control so there is a lot of freedom in what can be displayed. The default blog view consists of a single template that repeats for each post. It contains the title, author, date, a collection of tags, and the post content itself. The post content is stored as raw markdown, so it is first parsed through the markdown filter to generate HTML before being displayed. The about view is a simple container that renders a string of markdown in a similar fashion and displays it directly in the container without the need for any templates. The FAQ view uses a template like the blog view, containing the question as a simple string and the answer as markdown. In addition to the template, an ordered list is shown at the top of the view with anchors referencing each templated entry. Finally, the create view contains a static form with all the necessary fields to create a post. A preview toggle button is available that

will hide the text area and render its content as markdown. The submit button is disabled while the preview is active.

3.0 Functionality

3.1 Additional Libraries

Because the content of the site largely relies on markdown, Marked.js was included as a dependency to parse markdown and produce corresponding HTML. The repository for Marked.js is available here: <https://github.com/markedjs/marked>

Injecting the parsed markdown as HTML is inherently unsafe, so Angular-sanitize was included to strip out dangerous tags and directives when updating `ng-bind-html` directives. Angular-sanitize is available through the AngularJS API reference here: <https://docs.angularjs.org/api/ngSanitize>

Ng-tags-input provides an incredibly easy to implement, highly customisable, and intuitive tag input. It supports two-way binding to an array of tag strings which fit perfectly with the initial post JSON schema. It also allows the use of a custom handler when adding tags to validate, accept, or reject the incoming tag. This proved useful for prepending a hash to each tag if one wasn't already present. Ng-tags-input is available through its documentation here: <https://mbenford.github.io/ngTagsInput/>

The npm package Nodemon was installed to ease server development as it automatically restarts the underlying Node server upon a file changes. Nodemon is available through npm with `npm install nodemon`.

3.2 Client-Server Interaction

The server implements a RESTful API with all calls to it made under the `/api` path. The available endpoints and their respective methods are detailed in Table 2 of Section 1.0. All content for the client application is made available through these endpoints. When called, the server will attempt to open a JSON file containing the object to return and send it to the client, except for the `POST /posts` route where the server will prepend the incoming data to the file instead. Incoming posts through the `POST /posts` route are validated against a sample object to ensure that each field is present and contains the correct data type to prevent garbage that may cause rendering issues from entering the JSON file.

As per Table 1 of Section 1.0, the blog, about, and FAQ views pull their content through the server's REST API. The content is parsed from the JSON contained within the response and displayed. This is designed to allow changes to the site content without having to alter the physical page structure in HTML.

Rather than implement 404 middleware through Express, it was implemented in the client due to several issues trying to render a customisable error page within the `ng-view` area.

3.3 Operation

The following is an excerpt of the readme file included in the root of the project directory:

Note: You need to have node js installed, just use the latest 'Current' version found [here](#)

- *Install node js if you haven't already per above note*
- Download and extract the provided web_dev zip to a suitable location.
- Navigate using command line into the extracted directory e.g. `cd webdev-assignment-2`
- Install dependencies via: `npm install`. This only needs to be done the first time the folder is initialized
- Start the server with: `npm start`

If the commands are successful, the server should be available at `http://localhost:8080/`.

For development of server files, it is recommended that Nodemon, and included development dependency, is run in place of node. As per Section 3.1, Nodemon will automatically restart Node if any files change so that you don't have to. To start Nodemon with npm, call: `npm run dev`.