Theta Tau Theta Gamma Website Documentation

Honors Capstone WN23:
Rhea Chowdhury
Nick Johnson
Alexia Iatrides

ABSTRACT

This document serves to record and teach other brothers of Theta Tau Theta Gamma about how the website is structured so that it can be updated throughout the years without direct reliance on the subset of the brothers with web development experience. This document will be an attempt to explain everything involved in the website and will also include additional links to external documentation that might be of aid to the reader. This document might also abstract some of the machine setup from EECS280 - if there is something that is not working locally, refer to the setup tutorials on the public EECS280 website.

This document was created as part of the Winter 2023 Honors Capstone between Rhea Chowdhury #469, Nick Johnson #470, and Alexia Iatrides #488. This document represents the current state of the Theta Gamma website as of April 2023 and is encouraged to be updated to reflect the evolution of the website in future years.

HIGH LEVEL WEBSITE STRUCTURE

The Theta Tau Theta Gamma website is created using React (JavaScript) for all source code, content, and structure and styled with CSS Bootstrap. Currently, the website is static (no specific user data), but currently undergoing development for a brother login (transitioning the website to dynamic). All user data will be stored using SupaBase, an open-source PostgreSQL database. Currently, the website is hosted through GitHub Pages but will need to be hosted by another company once the webpage becomes dynamic.

VERSION CONTROL (GIT)

Git is the version control that is in use to distribute the source code of the website to multiple authors of the code. The website is hosted on the Theta Tau Theta Gamma GitHub found here. Git is a virtual repository (repo) of all of the code. A branch is a version of the repository that diverges from the main project (for instance: main where all production code goes and separate branches for each author of code or different revisions) and the user can switch between different branches via a checkout command.

Command line tools will be used for workflow (macOS, Windows).

To start working, you must create a local repo. First, create a folder (aka directory) that the project will reside in. To initialize the local repo via:

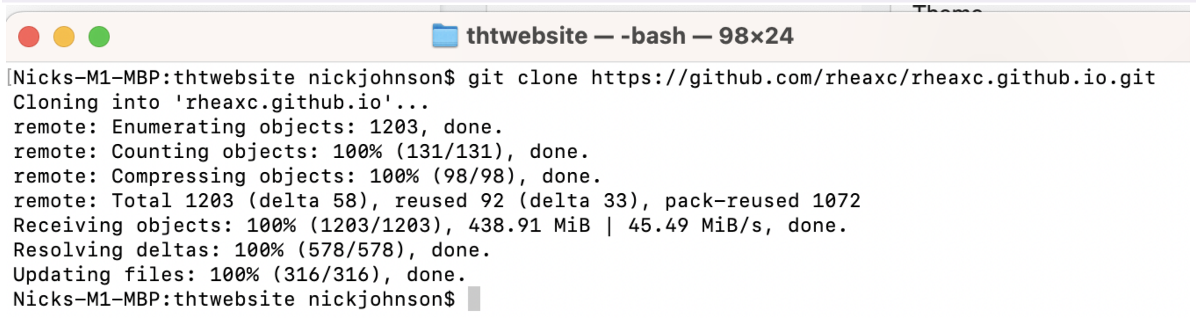
```
$ git init
```

This creates a local subdirectory .git which will contain repo files.

```
$ git status  // Check the status of local repo
```

You can now clone the GitHub repo by copying the HTTPS given on the webpage of the GitHub under "<> Code" and using the command line:

```
$ git clone <URL>
```



```
Nicks-M1-MBP:thtwebsite nickjohnson$ git clone https://github.com/rheaxc/rheaxc.github.io.git
Cloning into 'rheaxc.github.io'...
remote: Enumerating objects: 1203, done.
remote: Counting objects: 100% (131/131), done.
remote: Compressing objects: 100% (98/98), done.
remote: Total 1203 (delta 58), reused 92 (delta 33), pack-reused 1072
Receiving objects: 100% (1203/1203), 438.91 MiB | 45.49 MiB/s, done.
Resolving deltas: 100% (578/578), done.
Updating files: 100% (316/316), done.
Nicks-M1-MBP:thtwebsite nickjohnson$
```

To check that the local repo is connected to the remote repo (the GitHub), you will see the HTTPS of the remote repo:

```
$ git remote -v
```

Git Workflow and Commands:

When changes are made by other authors, to update local copies of the repo, use the command:

```
$ git pull
```

To see all of the branches in your local repository, use the command:

```
$ git branch
```

To see all of the branches in the remote repository, use the command:

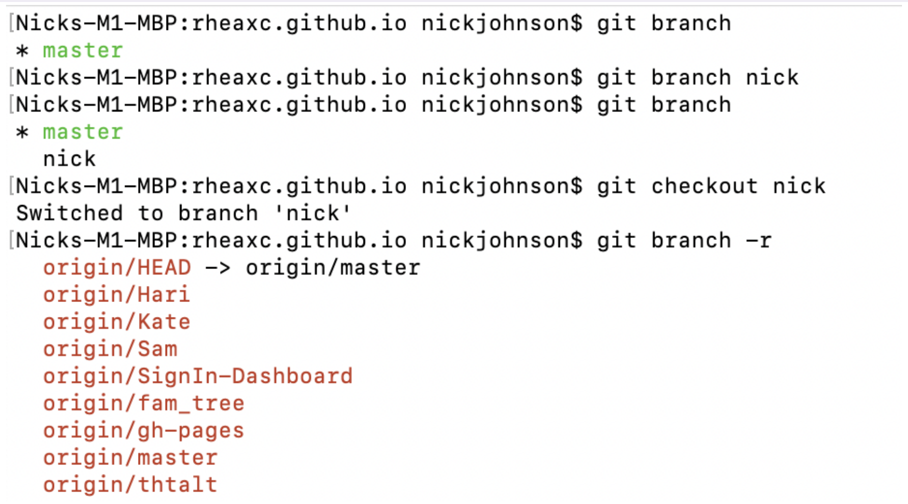
```
$ git branch -r
```

General rule of thumb is to never code on the main branch of the repo. To code on your own branch of the project, create a new branch using:

```
$ git branch <new_branch_name>
```

To navigate to your new branch,

```
$ git checkout <new_branch_name>
```



```
[Nicks-M1-MBP:rheaxc.github.io nickjohnson$ git branch
* master
[Nicks-M1-MBP:rheaxc.github.io nickjohnson$ git branch nick
[Nicks-M1-MBP:rheaxc.github.io nickjohnson$ git branch
* master
  nick
[Nicks-M1-MBP:rheaxc.github.io nickjohnson$ git checkout nick
Switched to branch 'nick'
[Nicks-M1-MBP:rheaxc.github.io nickjohnson$ git branch -r
  origin/HEAD -> origin/master
  origin/Hari
  origin/Kate
  origin/Sam
  origin/SignIn-Dashboard
  origin/fam_tree
  origin/gh-pages
  origin/master
  origin/thtalt
```

This will move you into your own branch where you can make your own changes to the code. To see the edits that you have made, use:

```
$ git status
```

This will return the save status of all of the files on the project. To “freeze” all of the changes that you have made to your code into the git, use:

```
$ git add
```

If you wish to add all of the files that you have changed, simply use:

```
$ git add .
```

Once you have frozen your changes, commit them to the local repository via:

```
$ git commit -m "Message"
```

Where “Message” is whatever text you want to show up in the GitHub history, this is typically a description of the changes/updates that were made. Finally, to push these changes to the remote repository for other authors to access, use:

```
$ git push -u origin
```

Sometimes git will reject the push, this typically comes from changes to the remote repository to the copy your local repository has. A way to fix this is to useL

```
$ git fetch
```

To see the difference between local and remote repos:

```
$ git status
```

Combine the local and remote commits with:

```
$ git rebase
```

Checking status again, push the commits to the remote repo as previously mentioned.

SET UP

Node.js is a runtime environment that the website uses - running code locally will not be possible without it. To check if Node.js is already installed locally (and the version if it is already installed):

```
$ Node -v
```

To download Node.js, follow the steps outlined directly on the developer's website. After installing Node.js, install the npm package manager - the standard module for Node.js. NPM is required to install any react features in order to run code locally. To install:

```
$ npm install -g npm
```

If there is any errors (terminal will likely tell you this):

```
$ sudo npm install -g npm
```

To verify successful installation (npm version):

```
$ npm -v
```

Now that Node and NPM are installed, set up the build dependencies (package.json) that are required to run the code:

```
$ npm install
```

If there are any errors with installing the dependencies:

```
$ npm install -force
```

Now that your environment is setup, code can be run locally via:

```
$ npm start
```

Which will load the web application at http://localhost:3000, Note that this instance of the webpage is hosted locally and cannot be accessed by anyone else (pseudo environment).

FIGMA

Figma is a collaborative design software the all front-end development of this website is prototyped and mocked-up on. It is a vector based program which allows for scaling across varying device screen sizes.

HTML

HTML, or HyperText Markup Language, is the standard structure for documents that will be displayed on the internet. For the case of a website, HTML tells the server how the webpage should be structured and styled for viewing. HTML creation is commonly assisted using CSS and a scripting language such as JavaScript - which will be further described in this paper.

HTML Elements

HTML consists of various elements such as body, headings, paragraphs, links, divisions, and images; a comprehensive list of HTML elements can be accessed via here. Each HTML element consists of 3 parts: the opening tag, content, and closing tag. It is worth noting that certain elements such as `<img>` must not have end tags. For example, a paragraph element (`<p>`) is denoted as:

```
<p> This is the content of the paragraph </p>
```

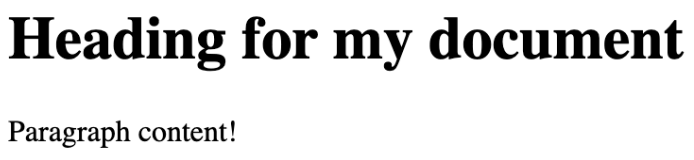
HTML elements can also be (and very often are) nested to create more complex pages. As an example: `<html>`, `<body>`, `<h1>`, and `<p>` elements are nested as:

```
<!DOCTYPE html>
<html>
<body>

<h1>Heading for my document</h1>
<p>Paragraph content!</p>

</body>
</html>
```

Where the `<html>` tag defines all of the content in between the opening and closing tags. Within the `<html>` element is a `<body>` element, which defines the body of the webpage. Continuously, within the `<body>` element contains both heading (`<h1>`) and paragraph (`<p>`) elements, with their own respective content. Such HTML above will be rendered on an webpage as shown in Figure 1 below:



Heading for my document

Paragraph content!

Figure 1. Sample HTML render demonstrating basic nested elements

Another HTML element that is important to the website is `<div>`, which defines a section of content within a page - acting as a container that allows all content between `<div>` and `</div>` to easily be manipulated (and styled with CSS). An example of an HTML `<div>` is below:

```
<div>
     <h1> Heading for my document </h1>
     <p> Paragraph content! </p>
</div>
```

Here, both the heading and paragraph elements are nested children within the `<div>` container element which allows for easy styling with CSS later.

HTML Attributes

HTML elements can also contain attributes which provide additional information about that element. An example of this is a hyperlink:

```
<a href="https://eecs.engin.umich.edu"> Text shown on webpage </a>
```

Where within the hyperlink element (`<a>`) element the `href` attribute specifies the URL of the linked webpage. Similarly, images can be embedded into the webpage via <img> element, where the `src` attribute specifies the path to the image as well as the `width` and `height` attributes declaring the display size of the image:

```
<img src="image.png" width="200" height="100">
```

Another attribute (`style`) allows for styling of the displayed content, and will be discussed in the CSS section of this document.

Two HTML attributes that are commonly used within the structure of the Theta Tau Theta Gamma website are `class` and `id`, which allow for naming of elements for easier reference when styling using CSS. Both elements essentially name elements - but class can exist for multiple elements meanwhile id can only exist for one. For example:

```
<div class="foo">
     <h1 id="heading_one"> Heading for this section! </h1>
     <p id="paragraph_one"> Paragraph content! </p>
</div>
<div class="bar">
     <h2> heading for this section </h2>
     <img src="image.png">
</div>
<div class = "foo">
    <h1 id="heading_two"> Heading for this section! </h1>
     <p id="paragraph_two"> Paragraph content! </p>
</div>
```

Here, the `class` “foo” refers to two instances of the `<div>` container, thus manipulation (such as styling) will be done to everything inside all of the instances of that class. Likewise, `id` can only refer to one instance of an element, thus all `id`s are unique and allow for manipulation of that specific element. Note that all elements can be named with `class` and `id` and it is good practice to name elements in a logical manner that allows for easier styling and maintenance.

NOTE: while `class` exists in HTML attributes, when scripting is done with React JS (as will be discussed later in this document) the `class` attribute turns into the `className` attribute as class refers to a reserved term in JavaScript (the data structure class). The `id` attribute remains unchanged between HTML and React JS.

There exists an extensive list of HTML attributes and can be accessed via here.

JAVASCRIPT/REACT

Javascript is a coding language. React is a framework that is built using the language Javascript that allows you to create a website. basically we use react for websites but its all javascript

While writing a static webpage in HTML is possible, it gets rather tedious and can be hard to maintain in addition to not being able to create dynamic pages. React JS leverages user's knowledge of HTML to build static and dynamic pages faster and easier while keeping a syntax similar to HTML.

React is a front-end infrastructure based in JavaScript. Through rendering, React/JavaScript source code is rendered into HTML. For purposes of this website, React is used as an object oriented language that the content of the webpage is written in. Such, the files that contain the structure and content of the website will now be .js instead of .html.

While syntax between HTML and React are similar, there are a few syntactic changes such as HTML attribute `class` becoming attribute `className` in React.

CSS

CSS, or Cascading Style Sheet, is a language used to style elements for the web page.

INLINE CSS:
Inline CSS is used directly in the element declaration using the `style` attribute (as described in the HTML section above) to directly style that element. For example:

```
<p style="color:black; background:grey"> Paragraph text </p>
```

Styles the paragraph element to have blue text and a grey background.

INTERNAL CSS:

Internal CSS can be used to style an entire HTML page and is declared within a <style> element inside the <head> element of the HTML page:

```
<!DOCTYPE html>
<html>
<head>
<style>
body {background-color: blue;}
h1   {color: yellow;}
</style>
</head>
<body>

<h1>Heading of Page</h1>

</body>
</html>
```

Where the body of this page is declared to have a blue background and the heading text to be yellow.

EXTERNAL CSS:
While the above methods of CSS are valid, both can prove hard to maintain as the pages get more advanced. External CSS is a process of abstracting the styling CSS into a file separate from the structure and content - specifically to this project: creating .css files that style corresponding .js files. To link these files together, the .css styling file (including the directory path if the files are in different directories) should be imported at the top of the .js file containing the page content. For example at the top of AboutUs.js the corresponding external CSS file is imported:

```
import './style/aboutus.css'
```

CSS files consist of rules for which styling declarations are made; each rule consists of a selector (the object to style) followed by a declaration (key-value pair of property:value) for the styled object. As an example, to style the color and alignment of a `<p>` element:

```
// example.js snippet
import 'style_example.css'

<p> this is the paragraph text </p>
```

The corresponding external styling.css file would be:

```
/* style_exmaple.css */
```

```
p {
   color: black;
   text-align: center;
}
```

It is worth noting that the word cascading in CSS means that the style that is applied to the parent element will also apply to all of the children elements nested (in React files, nests are indented) within the parent element.
As elements in the React files will be named through either `className` or `id` as covered earlier, referencing these objects in CSS will take the form of:

```
.object // selector for React className = "object"
#name // selector for React id = "name"
```

A simplified excerpt from 'src/components/homepage/AboutUs.js' is shown below:

```
<div className = "about-us-container">
      <div className = "about-us-item" id = "about-us-image-container">
            <img id="about-us-image" src='fullfrat.jpeg'>
      </div>
      <div className = "about-us-item" id = "about-us-text-container">
            <p id="about-us-text"> Text Goes Here! </p>
      </div>
</div>
```

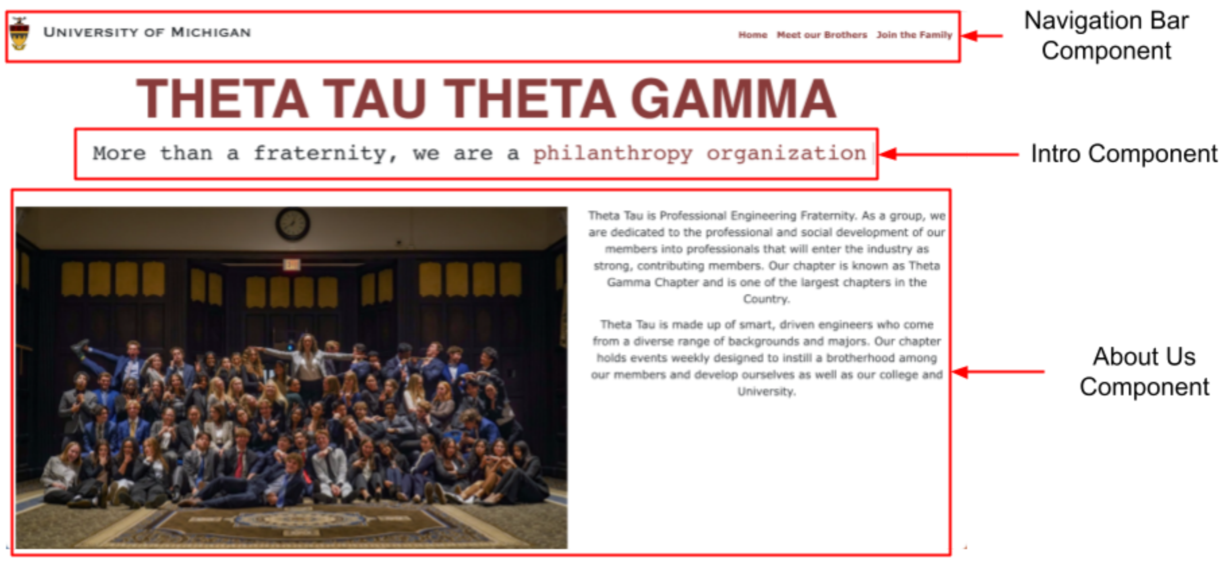
And a corresponding excerpt from 'src/components/homepage/style/aboutus.css' is shown below:

```
#about-us-image-container{
      width: 60%;
}
#about-us-image{
      max-width: 100%;
      max-height: 100%;
}
.about-us-item{
      padding-left: 1%;
      padding-right: 1%;
}
```

Breaking down the 'aboutus.css' file, `#about-us-image-container` sets the width of the `<div>` element with `id = "about-us-image-container"` to be 60% of the parent element, which is `<div className = "about-us-container">`. Second, `#about-us-image` sets the max-width and height of the `<img>` element with `id = "about-us-image"` to be 100% of the parent element's size (`<div className = "about-us-item" id ="about-us-image-container">`). Lastly, `.about-us-item` styles the two classes named "about-us-item". Note naming and indenting sections logically help to keep both the files organized.

WEBSITE STRUCTURE

Every single page on the Theta Gamma Website (Homepage, Members, and Rush) is made up of 'components' which can be compared to puzzle pieces that all fit together to form that page. This allows different sections (components) of a page to be developed separately and pieced together. For example, the 'Home' page of the website might appear to be like this:



THETA TAU THETA GAMMA

More than a fraternity, we are a philanthropy organization

Theta Tau is Professional Engineering Fraternity. As a group, we are dedicated to the professional and social development of our members into professionals that will enter the industry as strong, contributing members. Our chapter is known as Theta Gamma Chapter and is one of the largest chapters in the Country.

Theta Tau is made up of smart, driven engineers who come from a diverse range of backgrounds and majors. Our chapter holds events weekly designed to instill a brotherhood among our members and develop ourselves as well as our college and University.

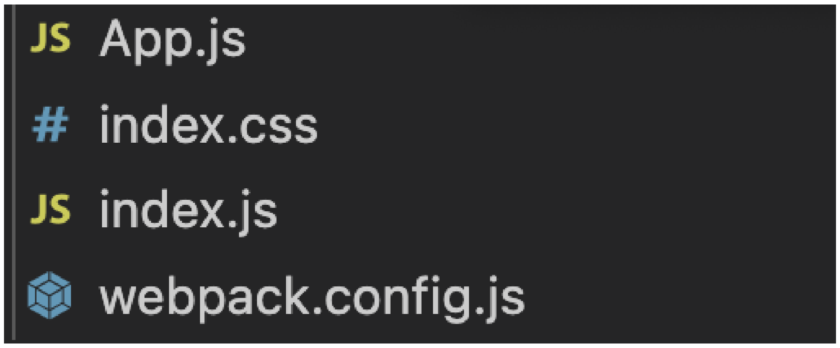
Where the 'Home' page is composed of several components that have their own React and CSS files, in addition to the overall HomePage.js and HomePage.css files that piece everything together.

Global Documents

`package.json` and `package-lock.json` are files that keep track of all dependencies (libraries that are used for the source code of the website) so that all developers can run the code without installing additional libraries. These files are automatically updated as new libraries/modules are used. *`CNAME`* is the file that contains the domain name of the webpage. `.gitignore` is the file that prevents unwanted files from being tracked into the remote repository by git.

SRC

This is the directory that holds all source content of the website (everything that will need to be updated through the years). The global documents within SRC are:



```
App.js
index.css
index.js
webpack.config.js
```

The most important of these SRC global documents is `App.js` which is the overall website that contains all pages within the website currently: Homepage, Members, and Rush. Using React Router the website renders the 3 different pages on the `Navbar` component and routes to either of those 3 pages depending on what the user selects.

SRC/Components

Within `src/components` is a directory for each of the 3 webpages currently on the website: Homepage, Members, and Rush. Each of these pages are made up of ‘component’ React and CSS files that return a `<div>`, which are then pieced together in the corresponding page file. For instance, `src/components/homepage` contains all of the component level React and CSS files which are then pieced together within `src/components/HomePage.js` and styled with `src/style/home.css`.

Some of the components that appear on all of the pages (navigation bar and footer) are at one level higher, in `src/components` and `src/style` since they appear on all of the 3 pages of the website.

BUILDING, RUNNING, DEBUGGING

Running the website locally is important for testing features before releasing them to the public website. To run the source code locally, first you must install Node.js, as outlined in the ‘Set Up’ section here.

To run locally, you must first download all the dependencies of the project (in package.json)”

```
$ npm install
```

Now you can run the website locally via

```
$ npm start
```

Which will load the web application at http://localhost:3000. Here the website can be interacted with exactly how the public will use it, all interactions reflect what the website will behave like.