Final Project Report (Taylor'd UI)

Michael Taylor, and Eamonn De'Leaster*

Waterford Institute of Technology,
Dept of Computing and Mathematics,
Cork Rd, Waterford City, Ireland
20064376@mail.wit.ie
http://www.wit.ie

Abstract. A complete User Interface framework designed for entry level users with a focus on a vocabulary that is easy to understand. Taylor'd UI will also introduce partials using web components, boilerplate code offering different themes and integrated command line interface.

Keywords: CSS, User Interface, framework, beginners, boilerplate, partials, themes

* Supervisor

Table of Contents

Glossary

CLI Command Line Interface. 8, 18

CSS Cascading stylesheets. 7, 8, 19, 22, 24, 31, 33

DRY Don't Repeat Yourself. 31, 33

em element. 23, 24

HSL Hue, Saturate, Lighten. 22

mixins a mixin is a class that contains methods for use by other classes without having to be the parent class of those other classes.. 19, 20, 24, 26, 31, 33, 36

px pixel. 23, 24

rem Root element. 23, 24, 29

SASS Syntactically Awesome Stylesheets. 19, 20, 33, 35, 36

SCSS Sassy Cascading Stylesheets. 19, 20, 33, 36

UI User Interface. 7, 10

List of Figures

List of Tables

Acknowledgements

I would like to thank ....

Introduction

Background

UI, and *CSS* frameworks are abundant and readily available and generally covered by permissible licence agreements [Arsenault, 2016]. This not only allows commercial use of the frameworks but ultimately encourages tie in to a particular framework. End users expect all websites to work across their devices, laptops, tablets and phones. This requirement of responsive web applications ensures that hand coding a solution is a daunting task for entry level developers.

Using a framework allows the user to speed up the initial mock-up process, they offer clarification on common *CSS* issues, and have wide browser support. Frameworks are good for responsive design, offer clean, and tidy code. There are disadvantages to using frameworks such as an abundance of unused code left over, this is seen more so in large front end frameworks such as Twitter Bootstrap. There is a slower learning curve with using frameworks, as the majority of work is done for them. All the user does is change small features such as colour, not encouraging them to learn through development.

Frameworks such as Twitter Bootstrap [et al, 2016b], and Zurb Foundation [et al, 2016a] offer complete solutions, with ready built code for forms, buttons, fluid layouts, and popovers. Skeleton [Gamache, 2016] is an example of the other end of the spectrum. Skeleton is a boilerplate for responsive, and mobile first development. It is designed to be light, and is built with less than 400 lines of code. Unlike Bootstrap or Foundation, skeleton is designed to be the users starting point, not their full solution.

This project has been built to offer the end user a complete solution designed with an entry level user in mind. The framework is non-opinionated unlike frameworks such as Bootstrap which is very opinionated about their design Guadia [2016]. This framework is designed to be the starting point which a user may then manipulate, and build upon.

Rather than building large complex websites with repeating code, one of the project goals is to allow a safe environment where a user can experiment with boilerplate templates, partials, and a *CLI*. Partials break up the html code into smaller more manageable fragments that can be used across multiple html files. The framework has been built with this in mind, creating classes, and id's that can be reused instead of having a bloated package with a lot of unused code [K, 2016].

Prebuilt code such as standard default themes are often referred to as boilerplate templates, a major benefit of this project is to enable the easy modification of such templates. By generating the templates for the user, it can allow them to concentrate on the aesthetics more so than the structure of the project. Using prebuilt themes will not suit every situation. The templates themselves can be expensive as can be seen on Themeforest [uouapps, 2017], and might not adhere to the W3C standards, or worse might offer little customisation [Weller, 2016].

Further attempts to incorporate Bootstrap into projects demonstrated the syntax as very unfriendly, and noticeably more difficult than hand coded *CSS*. Bootstrap syntax such as:

```
1 |    <div class="col-sm-4">
```

does not describe in any form that it would be displayed as a three column layout in the browser. Based on the snippet above, an entry

level user is unlikely to know how to change this from a three column layout to a single column layout.

Additionally while working as a web developer mentor at Coderdojo, students were observed to have similar experiences. Many were reluctant to learn frameworks such as Bootstrap as they were too confusing. Taylor'd UI could potentially be used in an educational environment to provide students a structured introduction to web development.

License

The application will be released under the MIT license following in the footsteps of the other frameworks researched for this project. The MIT license is permissive, allowing permissions such as commercial use, private use, distribution, and modification. With the MIT license, another user cannot claim the work as their own, derivatives are allowed as long as the original author is credited, and the original authors cannot be held liable.

User Manual

Introduction

The project is to build a full *UI* framework that is aimed towards an entry level user. The framework is to be non-opinionated, light weight, easy to understand, theme-able, and most importantly easy to learn.

The framework has minimal styling, the author has only added styling where it is needed, and that styling is kept to basic colours. This was to ensure the framework does not carry the author's design opinions. With been non-opinionated the framework is bare so that the end user is encouraged to not rely upon the built-in styling, but instead to use it as a starting point to build upon it. The framework is based on a responsive grid system of 12 grids, allowing the end user to build a seamless experience from desktops to mobile devices.

One of the aims is to make the library as small as possible to ensure it loads quickly on mobile devices and in situations where fast network connectivity is not available. This also ensures that the data used on mobile devices is not unnecessarily burdened.

To stop code from being repeated unnecessarily, code sections have been broken into partials to allow for the ability for the content to be broken up into manageable pieces, removing receptive code such as headers, and footers.

The framework will contain similar features from both Bootstrap, and Foundation as seen in table 1 such as tables, lists, breadcrumbs, and pagination. Features such as tooltips, and right to left language support are outside the scope of the project. The hope for the project is to be complete framework but having a smaller file size than both foundation, and Bootstrap borrowing concepts from Skeleton in keeping the framework light, and nimble.

The proposed framework will be built using the five key points mentioned below:

- For entry level users
- Theme-able
- Introduce the concepts of partials
- Have boilerplate templates for the user to use
- Have a command line interface to add a theme to a boilerplate template

Partials

Partials are reusable HTML snippets that you can be embed into a template file and render, this helps you modularise your development. By splitting up the code into partials, you are:

- Keeping your code clean and systematic
- Adhering to the DRY philosophy
- Creating reusability
- Aid in fragment caching

Not all modern browsers have full support for partials, and templates as seen on caniuse [Alexis Deveria, 2017]. In order to achieve partials, and templates, Taylor'd UI is using nunjucks a rich and powerful templating engine. If you are not comfortable in learning how to break the HTML into partials, and just want the HTML file to play around with, the file index.html can be found in the parent directory.

Before starting, from the terminal enter in the commands seen below, the first will install gulp, the second will install nunjucks:

```
npm install gulp-cli -g
npm install gulp-nunjucks-render --save-dev
```

A package.json file will then need to be created, this will configure the Node / NPM packages. Following this, create a gulp.js file, and place it in your main folder. Open the file and add the two following lines to the top.:

```
var gulp = require('gulp');
var nunjucksRender = require('gulp-nunjucks-render');
```

The gulp.js file tells gulp what to do. In this file, we will be adding in the tasks we want gulp to run.

Next, create a folder structure similar to layout structure as seen in figure 3. The templates folder is used for storing the Nunjucks partials, and any other Nunjucks files that will be added to files in the pages folder. The pages folder is used for storing files that will be compiled into HTML. Once they are compiled, they will be created in the blog folder.

Create a file and call it layout.nunjucks. The layout file will contain the boilerplate HTML code such as title, and links to external files such as the CSS, see below for an example.

```
<!-- layout.nunjucks -->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Simple Blog</title>
  <link rel="stylesheet" href="css/taylord.css" />
</head>
<body>

  <!-- You write code for this content block in another file -->
  {% block content %} {% endblock %}
</body>
</html>
```

In the pages folder, create a file called index.nunjucks. This file will ultimately be converted into index.html and placed in the blog folder. The index.nunjucks extends the layout.nunjucks file, this means it contains all the boilerplate code written in the layout file.

HTML code that is specific to index.nunjucks between the two block as seen below:

```
{% block content %}
<h1>This is our example heading</h1>
{% endblock %}
```

To generate the index.html file, a nunjucks task needs to be created that will do the conversion for us. Add the following code to the gulp.js file.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Simple Blog</title>
  <link rel="stylesheet" href="css/taylord.css">
</head>
<body>
  <h1>This is our example heading</h1>
</body>
</html>
```

Run gulp nunjucks from the terminal, you should see a new file called index.html created in the root folder. Open the file in a text editor. What you will find is something similar to below, a html file containing all the code from the layout file, and a heading that we added.

```
gulp.task('nunjucks', function() {
  // Gets .html and .nunjucks files in pages
  return gulp.src('blog/pages/**/*.+(html|nunjucks)')
  // Renders template with nunjucks
  .pipe(nunjucksRender({
      path: ['blog/templates']
    }))
  // output files in app folder
  .pipe(gulp.dest('blog'))
});
```

Moving on to the next stage, partials. Create a new file called nav.nunjucks in the partial folder. Create a navigation class like below:

```
  <nav>
    <ul>
      <li><a href="#" class="current">Home</a></li>
      <li><a href="#">About</a></li>
      <li><a href="#">Work</a></li>
      <li><a href="#">Blog</a></li>
      <li><a href="#">Contact</a></li>
    </ul>
  </nav>
```

Then add the following snippet to the layout file. This tells the layout.nunjucks file to include the navigation partial when it compiles. Run gulp nunjucks again.

```
{% include "partials/navigation.nunjucks" %}
```

Open the file in a text editor, and you should see something similar. Remember the placement if the include statement determines where the code is placed in the compiled HTML file.

```
\begin{lstlisting}[language=HTML]
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Simple Blog</title>
  <link rel="stylesheet" href="css/taylord.css">
</head>
<body>
  <nav>
    <ul>
      <li><a href="#" class="current">Home</a></li>
      <li><a href="#">About</a></li>
      <li><a href="#">Work</a></li>
      <li><a href="#">Blog</a></li>
      <li><a href="#">Contact</a></li>
    </ul>
  </nav>
  <h1>This is our example heading</h1>
</body>
</html>
```

You have now built your first template, and partial. You can now use the same navigation partial in other websites. Follow the exact same steps to create other partials like the footer, and any other content that you feel is either repeated or can be used elsewhere.

Themes

Talk about themes here

Boilerplate

talk about portfolio, blog and commerce website here

Command Line Interface

A *CLI*, also known as a Console User Interface is a text based interface that is a means of interacting with a computer program where the end user issues commands through an application such as the terminal in the form of successive lines of text.

Development

Technologies

CodeKit

To compile the *SCSS*subsection Codekit [Jones, 2017] was used. Although *SCSS* can be compiled from the command line or using a tool such as Grunt, using an application such as Codekit had extra features that would all have to be separate commands or even applications that made development easier.

SCSS

SCSS can be compiled by many different utilities such as Grunt. However, the decision was made to use Codekit [Jones, 2017]]. There are several reasons for this, it is a single application which compiles languages such as SASS, Less, Stylus, and CoffeeScript as well as offering tools such as built in local hosting that can be accessed from multiple devices, and automatic minifier all in a single package, and reduced the need to use many different applications to perform the same tasks performed by Codekit.

The initial idea was to build the framework code using *SASS*, however as development continued, *mixins* were found to be an issue when written in *SASS*, *SCSS mixins* could not be used with *SASS* code. It was decided that the use of *mixins* were more important in the development of the *CSS* than not. The *SASS* was converted into *SCSS* to continue development.

SCSS is an extension of the *CSS* syntax, which means that rules written in *CSS* can also be considered to be valid *SCSS*. *SCSS* uses

semi-colons and braces to break up the code of an element. Variables in *SCSS* are declared with $, and the assignment sign is :.

mixins are one of the most powerful features of *SASS*/ *SCSS*. They allow for efficient, and clean code repetitions as well as an easy way to adjust your code with ease. *mixins* are the *SCSS* equivalent of macros in other programming languages.

Nunjucks

Nunjucks is a rich, and powerful language used for templating. Nunjucks has features such as inheritance that other templating engines such as Handlebars do not. Gulp.js which is a task runner, is used alongside Nunjucks to remove the problem of repetition.

A template engine is a tool that allows a user to break HTML code into smaller pieces that can be reused across multiple HTML files. Templating engines allow developers to write dynamic code, which can change based on what is inserted into a particular variable. This is extremely useful in situations such as localisation where different phrasing might be required. This can be achieved by using translation module such as i18n [Spiegel, 2016]. Another major benefit to this is the ability to separate content from stylisation [Cohen, 2004]. Instead of using a application to develop the partials Gulp [Gulp.JS, 2017], and Nunjucks [Mozilla, 2017] will be used.

HTML

HTML is a markup language used in structuring, and presenting content on the Word Wide Web. HTML is the tool that the user will be using as the starting point of their development.

LaTeX

The LaTeX Project is a document preparation system for high-quality typesetting. The user uses markup tagging conventions to define the general structure of a document [Project, 2017]. With LaTeX a documents content can be quickly and easily developed then styled through the use of markup tagging conventions which define the general structure of a project.

Git

As the project will be large, with complicated parts, and to help in the development of the project, Git will be used for version control. Git works by creating snapshots of files. If there hasn't been any changes to specific file, Git will link it to a previous version, keeping the project fast, and lean [Torvalds, 2010].

Having versions of the project will show what changes have been made over time, and if needed allow the project to be roll backed to a previous version. If a regression bug is introduced into the framework, it can be rolled back to a previous version easily using git [Atlassian, 2017].

Framework

The Taylor'd UI framework is made up from the following components. Each of these components have been developed in separate partial files denoted by an underscore. The underscore is a keyword which is used to pass instructions to Codekit. A separate file called taylord.scss was created, in this file the @include statement was used to pull all the separate partial files into one file and that file is then compiled to *CSS*.

Variables

At the start of framework development, a file called _foundation.scss was created, in this file all the variables for the framework are defined. In this file, the foundation colour variable is set and using the *HSL* function called lighten.

HSL stands for Hue, Saturation, Lighten. Hue is a degree on the RGB colour wheel, this ranges from 0 to 360. 0 is red, 120 is green, and 240 is blue, Saturation is a percentage value; 0% means a shade of grey and 100% is the full colour. Lightness is also a percentage; 0% is black, 100% is white [W3schools, 2017].

A range of colours are created to be used in the framework. Other colour variables were added and then used to set the background colour, alert colours, and link colours.

```
1  $foundation-color: #000;
2  $foundation-color-2121: lighten($foundation-color,
3  13%);
4
5  h1, h2, h3, h4, h5, h6 {
6    color: $foundation-color-2121;
```

Font and Typography

In keeping with the framework being non opinionated, it was crucial to pick a font, and to have the typography nondescript so that the end user can change it to suit their needs. A clear indicator of a suitable font, is that it that it is readable when used in individual lines and also in large blocks of text.

Each candidate font was viewed on a range of devices including mobile, and desktop, thereby ensuring it was legible across multiple devices, and screen resolutions. Lastly the range of the font was looked at, did the font allow for non latin characters, accents, and symbols [Wordpress, 2012].

The font that passed these tests, and was visually appealing was Open Sans [Matteson, 2010]. Open Sans was commissioned by Google and designed by Steve Matteson. Open Sans was designed to have a friendly, but neutral appearance and is optimised for legibility across different mediums such as print, web and mobile.

The next step was to determine the font sizing and how to calculate it. Font sizing can be achieved using the three following methods; *px*, *em*, and *rem*. *px* was not looked at as early versions of Internet Explorer are not able to change the font size using browser functionality, a major usability issue.

The *em* technique alters the base font size on the body element by using a percentage [Laug, 2007]. This adapts the font so that 1em is equal to 10px, instead of the default 16px. To change the font size to the equivalent of 14px, the *em* needs to be 1.4em. The downside of using *em* to calculate the font sizing is that the font size compounds. This means that a list within a list isn't 14px but rather 20px. There is a work around where any child elements are declared to use 1em, but an entry level user would not know this.

With the advent of *CSS*3, *rem* was added, as previously mentioned, *em* sizing was relative to the font size of the parent whereas *rem* is relative to the root or html element. This means that a single font size can be defined for the html element, and all *rem* units will be a percentage of that base unit. Safari 5, Chrome, Firefox 3.6+, and even Internet Explorer 9 have support for *rem* units. Opera up to version 11.10, and early versions of Internet Explorer have yet to implement *rem* units. In order to display font on these browsers, a fallback *px* size is calculated using *mixins*.

Colour Scheme

When creating the colour scheme, it was important to keep design opinions to a minimum, to use colours that had more neutral tones, and that are not garish in appearance. It was also important to use colours that the end user would recognise that they were for instructional purposes, and for them to change in their own web development projects. The colour needed to look visually appealing when modified as well. This means if the hue, tone and vibrancy of the colour is modified, the resulting colour needed to be appealing as well.

Instead of researching colour theory and choosing the best colours, it was decided to use Google's Material Design colour palette, and choose from their wide range of colours [Google, 2017a]. The palette gave a variety of colours along with modifications of the colour such as different hues and saturations.

Tables

Tables in HTML should only be used for rendering data that naturally belongs in a grid based system. This is data where the data characterised is similar across a serveral objects. Tables should not be used for the layout of content in a website, divs should be used for this. The key to designing the tables was to demonstrate to the end user that the tables were for data, and not for layout.

Five table variations including table modifiers were created for the framework, ranging from default table to striped tables. Table modifiers take the colours that are used to dictate success or warning, and add them to a row in the table. The tables were designed to be responsive, this was achieved by taking the padding of table, and then dividing the $baseline height by a set value as seen below.

```
1    .table.table-condensed > thead > tr > th,
2    .table.table-condensed > tbody > tr > td {
3      padding-top: $baseline-size / 2.4; // 5px
4      padding-bottom: $baseline-size / 2.4;  // 5px
5      padding-left: $baseline-size / 1.5;  // 8px
6      padding-right: $baseline-size / 1.5; // 8px
7    }
```

Buttons

Buttons are an integral part of a framework. The styling, and the functionality of the buttons are key in the end users goal of using a website. If a button does not look like a button or if the styling of a button is overdone, the user can get confused, and not know how to proceed. The development of the buttons continued throughout the development of the framework. Originally, the buttons had round corners but this was removed in trying to keep the design non opinionated.

Five button types were designed at the start of the project. The default button, then large, and small buttons based off the default button, and lastly a pill shaped button.

Based off the default button type, six styles were created. Each of these button styles has a visual weighting to it such as the warning button. This button can tell the user to proceed with caution, it can also be used as a delete button. To create these buttons, *mixins* was used that takes the colour stored in the variable for each of these classes. The *mixins* also adds in the active and hover states, and calculates the colours to be used.

```
.button-default {
  @include button-version($button-default-color,
   $button-default-background);
  color: $foundation-color-a6a6;
  text-decoration: none;
  border: $foundation-color-e8e8;
  border-style: solid;
  border-width: thin;
  }
}
```

Panels

A panel is a component that allows you to outline a section of a web page. This enables you to view sections on your page as you add content to them, allowing you to place emphasis where you need it or removing all content from a section.

```
.panel-default {
  border-color: $default-border-bottom-color;

  .panel-title {
    @include panel-title($default-color-background,
    $default-panel-text, $default-border-bottom-color);
  }
}
```

Button Groups and Pagination

A button group is a series of buttons grouped together on a single line, this can be achieved by removing the margin attribute in CSS. Pagination is a series of numbers grouped together on a single line, this can be useful for when you have multiple pages in your website. As these two components can be used interchangeably, it was decided to include them together in the same partial.

To get these components working correctly, the > symbol was used, this allowed for only the direct children of an element to be selected, and modified. It will not affect any other element that is not a direct child of that element. In the code snippet below only the a element of unordered list item belonging to the class pagination gets a solid 1px border.

```
1 .pagination > ul > li > a {
2   border: 1px solid $foundation-color-e8e8;
3 }
```

Labels

Button labels should be kept as simple as possible. Long labels take longer to read, and can also take up large sections of valuable real estate on mobile web pages.

The button element illustrates a clickable button. The button element can be quite adaptable, elements such as images, text, headers, and even paragraphs can be in the button. The button element can also contain pseudo-elements such as ::before, and ::after.

There is a clear difference between the label element, and a button created with an input element. An input element serves a data field, this is user data that you intend to dispatch to a server. There are several types of input related to a button.

```
1 <input type="submit">, <input type="image">, <input type="file">,
2 <$input type="reset">, <input type="button">.
```

Navigation

The navigation component of the framework is a simple responsive navigation menu comprised of a non-list style that becomes a drop down menu when the screen size is less than 640px.

On a larger browser window size, the navigation is designed to stay at the top of the browser window when the user is scrolling down on a web page, this is achieved by making the navigation fixed to the web page. As screen real estate is a commodity on smaller browser window sizes such as mobile, the navigation bar changes to absolute. The navigation bar now stays at the top of web page, giving more screen real estate to the user.

```
///over 640px
header {
  background: $foundation-white;
  width: 100%;
  height: 80px;
  position: fixed;

///under 640px
  nav {
    ul {
      display: none;
      position: absolute;
      padding: 10px;
```

States

A state is an object that augments, and alters all other styles. For example, A message can be in a success or error state. States are commonly applied to the same element as a layout rule or applied to the same element as a base module class. In the frameworks, the states are used to indicate success in both alerts, and in a form.

```
.alert-success {
  color: darken($success-color, 15%);
  border-color: $success-color;
  background-color: lighten($success-color, 40%);
```

Grid

The grid is based on a 960px grid or 60rem in this case. The size is 60rem as it is based on the default font size of 1rem or 16px. Using that as the base of the calculation, a font size of 960px would be a *rem* value of 60. Modern desktops and laptops and mobile screens no longer tend to have resolutions below 960px, and for this reason the grid is set at this size as well as been evenly divisible in numerous ways as in in 2.

The 960 grid is adaptable to any layout or screen size. With using the 960 grid, a 12 column layout will utilised. The 12 column layout lends itself to the 960 grid as its also equally divisible, allowing for an odd number of columns all with even numbers as seen in figure 1 whereas using a 16 grid column layout, the same result is not easily achieved.

The framework has three breakpoints; desktop, tablet, and mobile. Based on the viewpoint, the columns expand or collapse in size.

```
1 $breakpoint-desktop: "screen and (min-width: 48rem)
2 and (max-width: 60rem)";
3 $breakpoint-tablet: "screen and (min-width: 30rem)
4 and (max-width: 47.9375rem)";
5 $breakpoint-mobile: "screen and (max-width: 29.9375rem)";
```

The media queries were developed using the rem mixin that was also used for font sizing, this allowed for the rem value and pixel value to be stored. For the columns, a media query for each target was included. Desktop, and tablet views have a rem, and pixel value. For mobile, percentages are used to better scale the content.

```
.col12 {
  @include rem(width, 960);

  @media #{$breakpoint-desktop} {
    @include rem(width, 960);
  }
  @media #{$breakpoint-tablet} {
    @include rem(width, 767);
  }
  @media #{$breakpoint-mobile} {
     width: 100%;
  }
}
```

Even though the framework is designed for 12 grids, when the media queries were designed, a query was not made for each column of the grid. Not every column has a media query attached to it. Instead the column around the missing column does the guess work for that column.

The framework has been designed to be a starting point, for better control the user needs to add media queries for each column. The framework is intended as a teaching tool for the user, adding in the queries for them, would not be beneficial.

Alerts

Alert notifications can be used to alert the user that something is about to happen or has happened, this can be that their username or password is incorrect, their login was successful, something went wrong while trying to load content, etc.

Five different alert types were created, each with their own significance. All the alerts are built using the same alert class, and through the use of the modifiers are changed into each alert type.

```
.alert {
  color: inherit;
  border: 0.5px solid transparent;
  display: block;
  padding: 1.5rem;
  background-color: $success-color;
  @include border-radius($border-radius);
```

Mixins

mixins are used throughout the development of this framework in an effort to keep the code *DRY*, (Don't Repeat Yourself). As seen in the example below, instead of having to write the border radius property for each browser, *mixins* were written to do this automatically. In the code, the *mixins* are called using the include statement @include border-radius. In the compiled *CSS*, the border radius property for each browser is added in automatically,

```
@mixin border-radius($radius) {
  border-radius: $radius;
  -webkit-border-radius: $radius;
  -moz-border-radius: $radius;
  -ms-border-radius: $radius;
}
```

Boilerplates, and Partials

A total of three boilerplate templates were made; a blog, portfolio, a product page. The templates were broken into partials so that there was no repetition of code, and for reusability. The boilerplates themselves are bare, and have the look of a wireframe, as they are intended to be. The boilerplates have been added to the framework as a learning tool, for the end user to manipulate with their own stylings, learning by doing.

To build the partials, Gulp, and Nunjucks was used in tandem along with a plugin called gulp-nunjucks-render [Kristijan Husak, 2017] . A gulp file was created that contained a script that is given the partial, and layout locations, and then compiles it into a html file using the command gulp nunjuks.

```
gulp.task('nunjucks', function() {
  // Gets .html and .nunjucks files in pages
  return gulp.src('boilerplate/blog/pages/**/*.+(html|nunjucks)')
  // Renders template with nunjucks
  .pipe(nunjucksRender({
      path: ['boilerplate/blog/templates']
    }))
  // output files in app folder
  .pipe(gulp.dest('boilerplate/blog'));
});
```

In an effort to remove any repetition, a loop was created in the layout file that repeated a certain partial for a set number of times, instead of having to recreate those elements multiple times.

```
{% for i in range(0, 3) -%}
  {% include "partials/portfolio.nunjucks" %}
{%- endfor %}
```

Development Issues

Originally the framework was to be developed using *SASS*, Several partials were developed, however to proceed further the use of *mixins* would be required. This was not possible with the existing *SASS*, and an effort to convert everything to *SCSS* was undertaken.

A bug within the button code surfaced which affected only certain buttons. This was an issue that plagued the developer throughout the development. The code for buttons were rewritten a few times to see where the issue was, and no matter what was written, the button was not clickable, this issue then effected the default button, along with the classes that used the default button as a base such as the primary button.

To figure out the cause of the issue, tools such as the Chrome Development tools were used. Using the inspect element feature, the button toggle state was used to force the element state as seen in figure 2. This allowed for each of the buttons states to be forced to run, while viewing the corresponding code to see where the error was.

It seemed that the button issue was caused by the style to be overwritten by a different partial. To get the buttons to have their different states, the active, hover, and focus elements had to be hard coded into the button. In doing this, it didn't matter what other styles might override the values as the elements were hard coded.

A minor issue was trying to keep the code *DRY* both the *SASS*. and the compiled *CSS* was viewed to see if there was repeating code. When there was repeating code, *mixins* were used where possible. Although there were sections of repeating code that converting to *mixins* was not possible.

Originally, the partials were going to be developed using Web Components [Components, 2017], and Polymer [Google, 2017b]. The

research had been done on these technologies, and how to use them. The development had started on the partials using these technologies but as the first partials were created, another student had viewing the work and asked *"Is this not too complicated for your user?"*, and they were correct. The framework has been designed for the entry level user to understand, and use. Using the latest in web standards is likely to confuse them more.

The below code snippet shows how a partial would work in using polymer. A template is made in a separate file and then called into the index file. The template is broken into different sections, the template section, and then the polymer section. Polymer is used to ensure that the template file is displayed in all browsers as the template tag is bot support by all browsers currently.

```
\\example-app.html
dom-module id="example-app">
  <template>
    <style>
      :host {
        display: block;
      }
    </style>
    <h2>Hello there[[prop1]]</h2>
  </template>

  <script>
    Polymer({

      is: 'example-app',

      properties: {
        prop1: {
          type: String,
          value: 'example-app',
        },
      },

    });
  </script>
</dom-module>
```

```
27
28 //index.html
29 <link rel="import" href="../iron-component-page/
30 iron-component-page.html">
31 <body>
32    <iron-component-page src="example-app.html">
33    </iron-component-page>
34 </body>
```

themes were difficult to think off.

One of the biggest obstacles in the development of this framework was deciding on what components were critical to the framework. Then what elements would the end user want. And finally, what elements does the developer think will be useful for the end user, elements that they need to understand without overcomplicating the framework, and or doing the work of the end user for them.

On of the biggest issues in the development of the framework was creating the grid layout, and media queries. The first major issue was the syntax. The breakpoints had been declared in a different partial, this lead to errors in the compiler. The compiler was looking for specific keywords after @media, which it could not interpret.

To solve this issue, the query had to be writing using different syntax. The variables for the breakpoints had to be treated as an ID. The compiler also would not accept brackets to frame the argument, curly braces had to be used instead. To reach this conclusion, a lot of trial and error was done. This involved making small changes to the @media argument, viewing the error, viewing *SASS* blogs to read what the error was in full, repeating this pattern until the code compiled with no errors.

```
1 //original
2 @media ($breakpoint-mobile)
3
4 //new method
5 @media #{$breakpoint-mobile}
```

Project Plan

Engineering Release One (January, and February):

- Iteration One: (9th of January - 23rd of January)
  - Met with Eamonn to discuss the best approach for the development in this semester
  - Changed how partials would be created, and displayed from a static site generator to using web components, and polymer to display the websites across browsers that don't support the web components import element
- Iteration Two: (23rd of January - 6th of February)
  - Started development of the framework. Created the file structure, and a partial called base that will contain all the variables of the framework
  - Decided on the colours and fonts that will be used in the framework
  - Created the body partial that contains a generic layout for a html file
- Iteration Three: (6th of February to 20th of February)
  - Created the button partial
  - Created the label partial to be used with the button partial
  - Created general layout of semester two document, broke the document into two sections; user manual, and development

Engineering Release Two (March):

- Iteration Four: (20th of February - 6th of March)
  - Rewrote all the *SASS* files to *SCSS*
  - Developed button *mixins* that would calculate all the button sizes
  - Created Alerts partial

  - Created States partial
  - Created Panel partial
- Iteration Five: (6th of March - 20th of March)
  - Created Table partial
  - Created Form partial

Engineering Release Three (April):

- Iteration Six: (20th of March - 3rd of April)
  - Started work on the development section of the document, broke the development section into smaller sections for each partial.
  - Created Button Groups and Pagination partial
  - Created Navigation partial
  - Designed the poster that would accompany the project.
- Iteration Seven: (3rd of April - 17th of April)
  - Met with Eamonn, decided on final structure of report
  - Added in Engineering Releases section
  - Finalised all the sections of the report, reworded sections, ready for submission.

Future Work

The original plan of the framework was to create a documentation website showcasing the framework, and detailing everything about the framework with code snippets, and examples of usage. When the framework was being developed, a kitchen sink website was created first to ensure that the development was working as it should. Due to time constraints, a finished website fully demonstrating the framework hasn't been developed, currently a single page website used for testing has been built with all the features of the framework akin to a kitchen sink example.

For the continuation of the project, a website will be built using the framework, and have documentation on each of the core functionality of the framework. In continuing the developing the framework, the following features would be added to the framework to make it more robust, and eliminate the need to write extra CSS:

- Badges to show unread content or to be used to display notifications
- Breadcrumbs to show the user where they are on a website
- Tabs for tabbed navigation
- Off canvas sidebar that slides in and out of a page, ideal for mobile design
- Modal to create modal dialog boxes
- Inverse to reverse the style of any component for light or dark backgrounds

In addition to CSS, jQuery features to be added that would help the framework work more seamlessly such as making links active in the navigation bar, ensuring that the footer section always stays

at the bottom of the page, regardless of page height, adding close options to the alerts, et al.

Conclusions

time management issues project management issues wanting to do more

Appendix

Fig. 1: 12 grid layout with the 960 grid

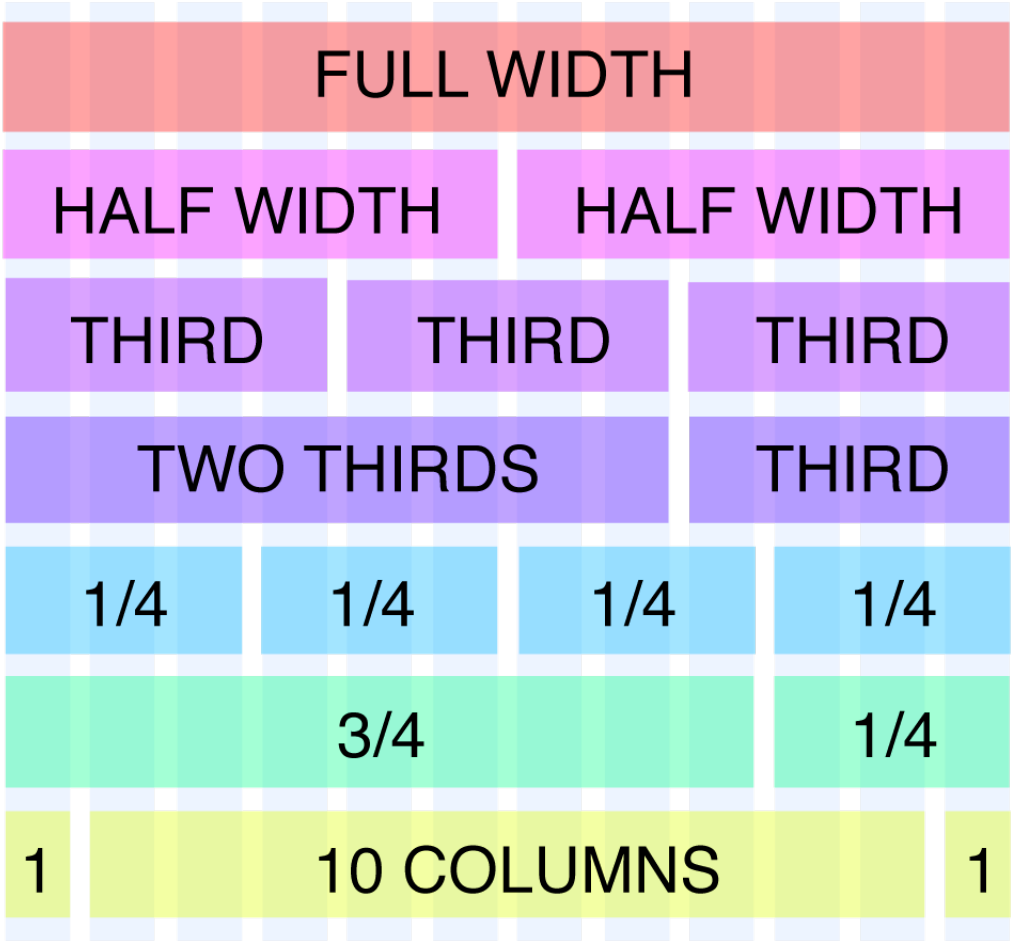

Fig. 2: Chrome Developer Tools

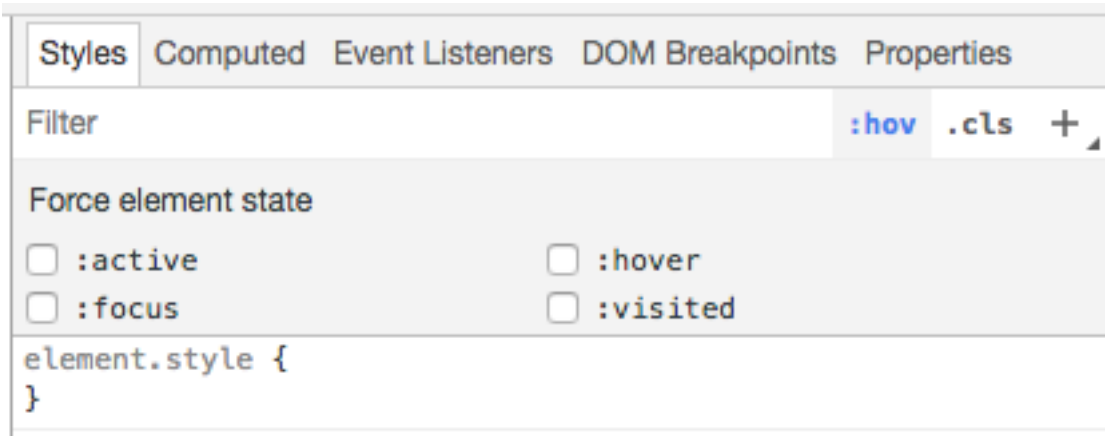

Fig. 3: Nunjucks File Structure

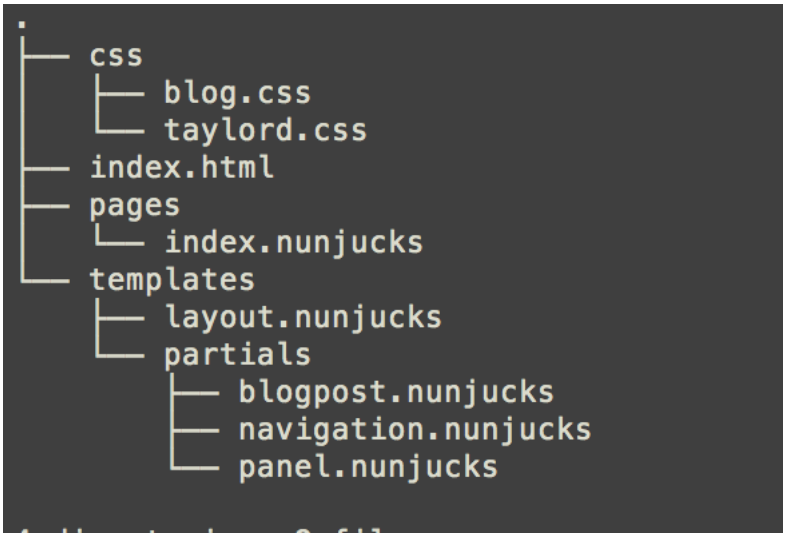

Table 1: Framework features

	Bootstrap	Foundation	Skeleton
Alerts	Yes	Yes	No
Accordion	Yes	Yes	No
Badges	No	Yes	No
Breadcrumbs	Yes	Yes	No
Buttons	Yes	Yes	Yes
Carousel	Yes	Yes	No
Dropdown	Yes	Yes	No
Forms	Yes	Yes	Yes
Form Validation	Yes	Yes	No
Grid	Yes	Yes	Yes
Icons	No	Yes	No
Labels	Yes	Yes	No
Lists	Yes	Yes	Yes
Media Object	Yes	Yes	Yes
Modals	Yes	Yes	No
Navigation	Yes	Yes	No
Pagination	Yes	Yes	No
Panels	Yes	Yes	No
Popovers	Yes	Yes	No
Print Styles	Yes	Yes	Yes
Progress Bar	Yes	Yes	No
Responsive Media	No	Yes	No
Right to Left	No	Yes	No
Scrollspy	Yes	Yes	No
Tables	Yes	Yes	Yes
Tabs	Yes	Yes	No
Thumbnails	Yes	Yes	No
Tooltips	Yes	Yes	No
Typeahead	No	No	No
Typography	Yes	Yes	Yes
Video Scaling	Yes	Yes	No

Table 2: Column Layout

Layout	Number of Columns
2 x 480	2 columns
3 x 320	3 columns
4 x 240	4 columns
5 x 192	5 columns
6 x 160	6 columns
8 x 120	8 columns
10 x 96	10 columns
12 x 80	12 columns
16 x 60	16 columns
20 x 48	20 columns
24 x 50	24 columns
30 x 32	30 columns

Bibliography

Lennart Schoors Alexis Deveria. *HTML templates* , 2017. Available at: http://caniuse.com/#feat=template [Accessed: 20/04/2017].

Cody Arsenault. *Top 10 Front-End Frameworks of 2016*, 2016. Available at: https://www.keycdn.com/blog/front-end-frameworks/ [Accessed: 03/12/2016].

Atlassian. *Undoing Changes* , 2017. Available at: https://www.atlassian.com/git/tutorials/undoing-changes [Accessed: 20/04/2017].

Michael Cohen. *Separation: The Web Designer's Dilemma* , 2004. Available at: https://alistapart.com/article/separationdilemma [Accessed: 20/04/2017].

Web Components. *Discuss and share web components* , 2017. Available at: https://www.webcomponents.org [Accessed: 05/04/2017].

Alexi Sellier et al. *Getting Started*, 2016a. Available at: http://lesscss.org [Accessed: 26/11/2016].

Hampton Catlin et al. *CSS with superpowers*, 2016b. Available at: http://SASS-lang.com [Accessed: 26/11/2016].

Dave Gamache. *A dead simple, responsive boilerplate.* , 2016. Available at: http://getskeleton.com [Accessed: 26/11/2016].

Google. *Style* , 2017a. Available at: https://material.io/guidelines/style/color.html# [Accessed: 05/02/2017].

Google. *Polymer 2.0* , 2017b. Available at: https://www.polymer-project.org [Accessed: 05/04/2017].

Kemie Guadia. *CSS Frameworks- comparing Bootstrap alternatives*, 2016. Available at: http://www.monolinea.com/css-frameworks-comparison/ [Accessed: 08/12/2016].

Gulp.JS. *Automate and enhance your workflow* , 2017. Available at: http://gulpjs.com [Accessed: 18/04/2017].

Bryan Jones. *CodeKit* , 2017. Available at: https://codekitapp.com [Accessed: 04/01/2017].

Karol K. *The Bootstrap Framework Controversy ... Should You Use It or Not?*, 2016. Available at: http://www.htmlcenter.com/blog/the-bootstrap-framework-controversy-should-you-use-it-or-not/ [Accessed: 08/12/2016].

Carlos G. Limardo Kristijan Husak. *gulp-render-nunjucks* , 2017. Available at: https://www.npmjs.com/package/gulp-render-nunjucks [Accessed: 18/04/2017].

Gun Laug. *em font-resizing bug in IE5 - IE7* , 2007. Available at: http://www.gunlaug.no/contents/wd_additions_13.html [Accessed: 25/01/2017].

Steve Matteson. *BlogU Multi-Author Magazine with Front-end User Page* , 2010. Available at: https://fonts.google.com/specimen/Open+Sans [Accessed: 25/01/2017].

Mozilla. *A rich and powerful templating language for JavaScript.* , 2017. Available at: https://mozilla.github.io/nunjucks/ [Accessed: 18/04/2017].

LaTeX3 Project. *gulp-render-nunjucks* , 2017. Available at: https://www.latex-project.org [Accessed: 04/01/2017].

Marcus Spiegel. *i18n* , 2016. Available at: https://www.npmjs.com/package/i18n [Accessed: 20/04/2017].

Linus Torvalds. *Git –everything-is-local* , 2010. Available at: https://git-scm.com [Accessed: 03/01/2017].

uouapps. *BlogU Multi-Author Magazine with Front-end User Page* , 2017. Available at: https://themeforest.net/item/blogu-multiauthor-magazine-with-frontend-user-page/14822417?s_rank=2 [Accessed: 19/04/2017].

W3schools. *Colors HSL* , 2017. Available at: https://www.w3schools.com/colors/colors_hsl.asp [Accessed: 20/04/2017].

Nathan B. Weller. *Custom vs Pre-made WordPress Themes A Look at the Pros and Cons*, 2016. Available at: http://wplift.com/custom-vs-pre-made-themes [Accessed: 08/12/2016].

Wordpress. *Open Sans, how do we love thee? Let us count the ways.* , 2012. Available at: https://en.blog.wordpress.com/2012/10/09/open-sans-how-do-we-love-thee-let-us-count-the-ways/ [Accessed: 03/02/2017].