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, templates offering different prebuilt websites along with different themes that can be used.

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

[*] Supervisor

Acknowledgements

Table of Contents

Glossary

CLI Command Line Interface. 59

CSS Cascading stylesheets. 9, 10, 30, 33, 34, 36, 39, 41, 44, 48, 52, 58, 59

DRY Don’t Repeat Yourself. 29, 48, 52

em element. 40, 41

HSL Hue, Saturate, Lighten. 39

HTML HyperText Markup Language. 10, 29, 30, 33, 37, 41, 42, 49, 50

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.. 36, 37, 41, 43, 48, 52, 56

rem Node Package Manager. 59

px pixel. 40, 41

rem Root element. 40, 41, 46

SASS Syntactically Awesome Stylesheets. 36, 37, 41, 52, 54, 56

SCSS Sassy Cascading Stylesheets. 36, 37, 52, 56

UI User Interface. 9, 13

List of Figures

List of Tables

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, users tend to make only minor changes such as modifications to colour.

Frameworks such as Twitter Bootstrap [Catlin, Eppstein, and Weizenbaum, 2016], and Zurb Foundation [Sellier, Page, Bointon, Jurcovicova, Dean, and Mikhailov, 2016] offer complete solutions, with ready built code for forms, buttons, fluid layouts, and popovers. Skeleton 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 [Gamache, 2016]. Unlike Bootstrap or Foundation, Skeleton is designed to be the users starting point, not their full solution.

Taylor'd UI has been designed and built with the intension of offering a complete solution aimed at an entry level user. 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 templates, partials. 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 [Król, 2016].

Prebuilt code such as standard default themes are often referred to as a template. 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:

```
    <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.

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. Taylor'd UI has five major aspects to it:

- For entry level users
- Theme-able
- Introduce the concepts of partials
- Have templates for the user to manipulate
- Install framework from the command line

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.

The link to this project can be found at:
https://github.com/MikeZTaylor/Taylord-UI

Taylor’d UI

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

The vocabulary used in Taylor’d UI has been designed to be user friendly so that when a user reads the syntax, they should have an idea of what the end result will look like without writing any code. The vocabulary follows english terminology, for example a pill shaped buttons syntax is:

```html
<button class="pill-shaped-button">Button</button>
```

Taylor’d UI has minimal styling, styling has only been added where needed. That styling is kept to basic colours. This was to ensure the framework does not carry the design opinions of the developer. Being non-opinionated, Taylor’d UI is intensionally 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.

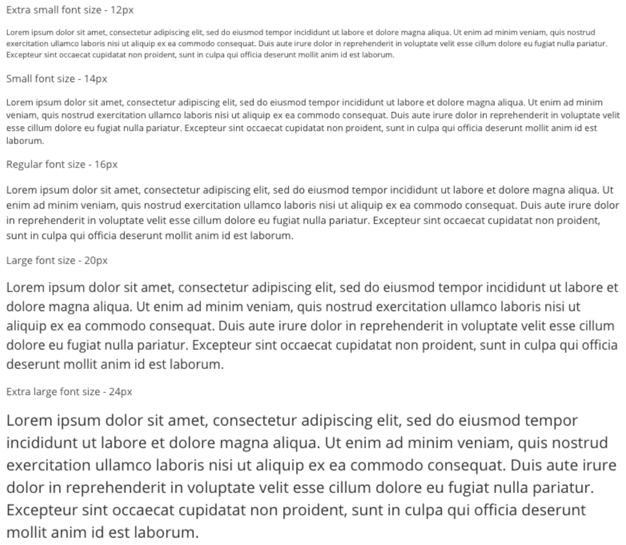
One of the aims is to make the library as small as possible to ensure it loads quickly on mobile devices and in situations where fast network connectivity is not available. This also ensures that the data used on mobile devices is not unnecessarily burdened.

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.

Taylor’d UI is made up of the following features; typography, buttons, alerts, tables, panels, jumbotron, button groups, forms, and a responsive 12 grid system.

Typography

The typography used in Taylor'd UI is Open Sans. The default font size throughout the document is 16. There are text modifiers such as extra-small-p, and extra-large-p that decrease or increase the font size as seen on Figure. 1 on page 14.



Extra small font size - 12px

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Small font size - 14px

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Regular font size - 16px

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Large font size - 20px

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Extra large font size - 24px

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Fig. 1: Font size range

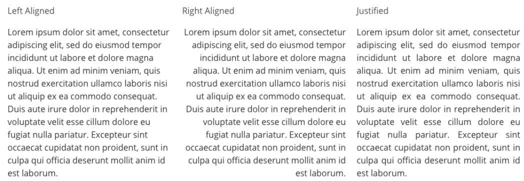
These classes can be called using the following methods:

```html
<p class="extra-small-p">Text</p>
<p class="small-p">Text</p>
<p class="large-p">Text</p>
<p class="extra-large-p">Text</p>
```

The text can be aligned; left aligned, right aligned or justified as seen on Figure. 2 on page 15. These classes can be called directly from the paragraph tag.

These classes can be called using the following methods:

```html
<p class="left-align-text"></p>
<p class="right-align-text"></p>
<p class="justified-text"></p>
```



Left Aligned

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Right Aligned

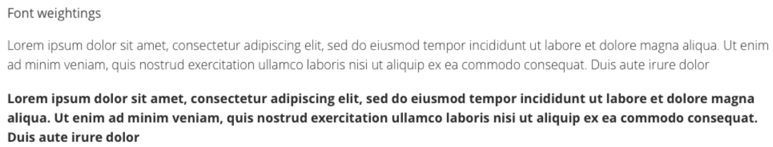
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Justified

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Fig. 2: Text Alignments

There are four different types of font weightings as seen on Figure. 3 on page 15. These font weightings can be used on paragraph, label, and heading texts.



Font weightings

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor

Fig. 3: Font Weighting

These weightings range from thin to bold. These classes can be called using the following methods:

```
<p class="thin-p">Text</p>
<p class="bold-p">Text</p>
```

The default font weighting for the paragraph text is 400. To change the weighting of the font, you can change the number value at the end of the variable. To save confusion, it would also be a good idea to change the variable name as seen below.

```
$font-weight-400: 400; //default
```

```
$font-weight-400: 700; //bolder
```

Buttons

There are four button types in Taylor'd UI as seen in Figure. 4 on page 16. These button types are styled in a similar way to how a browser would display a button.

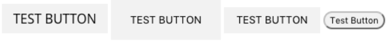


Fig. 4: Generic Button Types

These classes can be called using the following methods:

```
 <button class="button">Test Button</button>
<button class="pill-shaped-button">Test Button</button>
```

Additionally, a button can be styled using modifier classes. These modifiers are based on the primary colours as seen in Figure. 5 on page 16. These colours are colours that a user would have seen in other applications used for alerts such warning, and success actions.

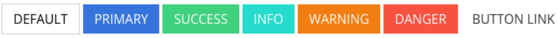


Fig. 5: Button Modifiers

These classes can be called using the following methods:

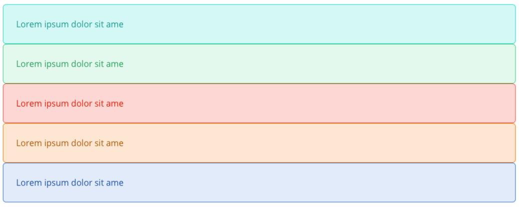
```
<button class="button default-button">default</button>
<button class="button primary-button">primary</button>
<button class="button success-button">success</button>
<button class="button info-button">Info</button>
<button class="button warning-button">warning</button>
<button class="button danger-button">danger</button>
<button class="button link-button">button Link</button>
```

The button modifiers can be used with the generic button classes. An example on how to call these classes:

```
<button class="large-button info-button"></button>
<button class="pill-shaped-button warning-button"></button>
```

Alerts

Alerts provide user feedback based on certain actions performed by them. An alert is built using the *.alert* class, and styled with modifier classes such as seen in Figure. 6 on page 17.



Lorem ipsum dolor sit ame

Lorem ipsum dolor sit ame

Lorem ipsum dolor sit ame

Lorem ipsum dolor sit ame

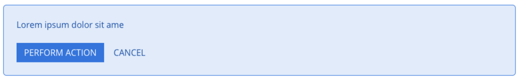
Lorem ipsum dolor sit ame

Fig. 6: Alerts

An example on how these classes can be called using the following methods:

```
<div class="alert info-alert">
<p>Lorem ipsum dolor sit ame</p>
<div class="alert success-alert">
<p>Lorem ipsum dolor sit ame</p>
so please pay attention.</p></div>
```

An alert class can be built upon by adding buttons, see Figure. 7 on page 17 as well as links.



Lorem ipsum dolor sit ame

Fig. 7: Alert Modifiers

An example on how these classes can be called using the following methods:

```
<div class="alert alert-primary">
<p>Lorem ipsum dolor sit ame</p>
<button class="button button-primary">Do Some Action</button>
 <button class="button button-link">Cancel</button>
```

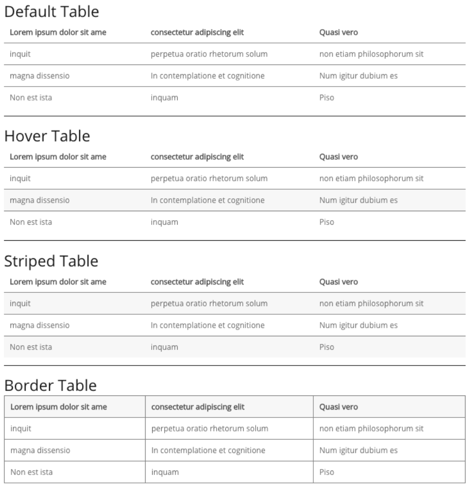
Tables

A table allow web authors to arrange data such as text, images, links, even other tables into rows and columns of cells. The tables are created using the table tag in which the *tr* tag is used to create table rows and td tag is used to create data cells.

A default table was created for Taylor'd UI. The default table is simple, with clear emphasis on the headings of the table.

```
<table class="table">
  <thead>
    <tr>
      <th>Lorem ipsum dolor sit ame</th>
      <th>consectetur adipiscing elit</th>
      <th>Quasi vero</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>inquit</td>
      <td>perpetua oratio rhetorum solum</td>
      <td>non etiam philosophorum sit</td>
    </tr>
    <tr>
      <td>magna dissensio</td>
      <td>In contemplatione et cognitione</td>
      <td>Num igitur dubium es</td>
    </tr>
    <tr>
      <td>Non est ista</td>
      <td>inquam</td>
      <td>Piso</td>
    </tr>
  </tbody>
</table>
```

There are four modifiers see Figure. 8 on page 19 that can be added to the table class for a different look as seen below. The other modifiers are hover-on-table, bordered-table.



Default Table

Lorem ipsum dolor sit ame	consectetur adipiscing elit	Quasi vero
inquit	perpetua oratio rhetorum solum	non etiam philosophorum sit
magna dissensio	In contemplatione et cognitione	Num igitur dubium es
Non est ista	inquam	Piso

Hover Table

Lorem ipsum dolor sit ame	consectetur adipiscing elit	Quasi vero
inquit	perpetua oratio rhetorum solum	non etiam philosophorum sit
magna dissensio	In contemplatione et cognitione	Num igitur dubium es
Non est ista	inquam	Piso

Striped Table

Lorem ipsum dolor sit ame	consectetur adipiscing elit	Quasi vero
inquit	perpetua oratio rhetorum solum	non etiam philosophorum sit
magna dissensio	In contemplatione et cognitione	Num igitur dubium es
Non est ista	inquam	Piso

Border Table

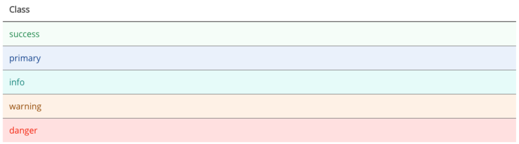
Lorem ipsum dolor sit ame	consectetur adipiscing elit	Quasi vero
inquit	perpetua oratio rhetorum solum	non etiam philosophorum sit
magna dissensio	In contemplatione et cognitione	Num igitur dubium es
Non est ista	inquam	Piso

Fig. 8: Tables with Modifiers

Below is an example of how to call one of the modifiers.

```html
<table class="table striped-table">
  <thead>
    <tr>
      <th>Lorem ipsum dolor sit ame</th>
      <th>consectetur adipiscing elit</th>
      <th>Quasi vero</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>inquit</td>
      <td>perpetua oratio rhetorum solum</td>
      <td>non etiam philosophorum sit</td>
    </tr>
    <tr>
      <td>magna dissensio</td>
      <td>In contemplatione et cognitione</td>
      <td>Num igitur dubium es</td>
    </tr>
    <tr>
      <td>Non est ista</td>
      <td>inquam</td>
      <td>Piso</td>
    </tr>
</tbody></table>
```

Modifiers from other components such as alerts can be used on table elements, as seen on Figure. 9 on page 20.



Class
success
primary
info
warning
danger

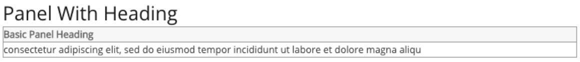
Fig. 9: Table with other Class Modifiers

The code snippet below shows you how to call the different modifiers.

```html
<table class="table">
  <thead>
    <tr>
      <th>Modifiers</th>
    </tr>
  </thead>
  <tbody>
    <tr class="is-success">
      <td>success</td>
    </tr>
    <tr class="is-primary">
      <td>primary</td>
    </tr>
    <tr class="is-info">
      <td>info</td>
    </tr>
    <tr class="is-warning">
      <td>warning</td>
    </tr>
    <tr class="is-danger">
      <td>danger</td>
    </tr>
  </tbody>
</table>
```

Panels

A panel in Taylor'd UI is a bordered box with some padding around its content. The basic panel is just that, see Figure. 10 on page 21. Other attributes such as heading tags can be added to a panel as seen below:



Panel With Heading

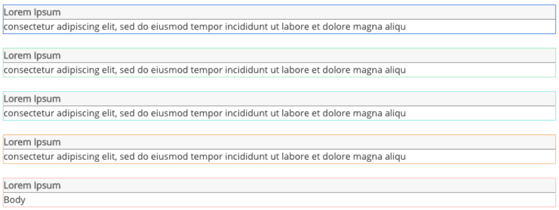
Basic Panel Heading

consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqu

Fig. 10: Default Panel

```
<div class="panel default-panel">
  <div class="panel-title">Lorem Ipsumn
  </div>
  <div class="panel-body"> consectetur adipiscing elit,
  sed do eiusmod tempor incididunt ut labore et dolore magna aliqu
  </div>
</div>
```

Modifiers from other components such as alerts see Figure. 11 on page 21 can be added to a panel. The code snippet below shows you one example:



Lorem Ipsum

consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqu

Lorem Ipsum

consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqu

Lorem Ipsum

consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqu

Lorem Ipsum

consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqu

Lorem Ipsum

Body

Fig. 11: Panel with Modifiers

```
<div class="panel panel-info">
  <div class="panel-title">orem Ipsum</div>
  <div class="panel-body"> consectetur adipiscing elit,
  sed do eiusmod tempor incididunt ut labore et dolore magna aliqu
  </div>
```

```
</div>
```

Labels

Labels are built with button-label class, the default label does not need the button class, as seen below:

```
<span class="button-label">Default Label</span>
```

Modifiers can be added to the label see Figure. 12 on page 22 for more impact.

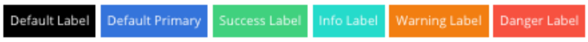


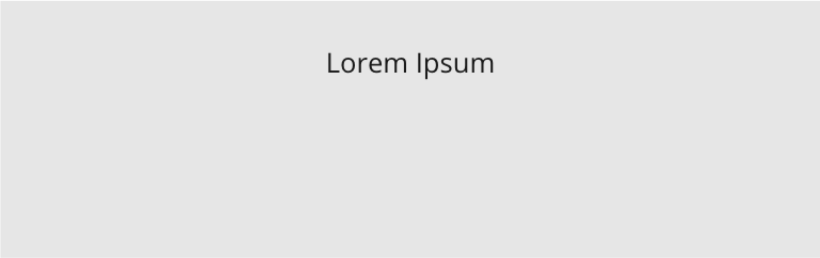
Fig. 12: Labels

An example on how these classes can be called using the following methods:

```
<span class="button-label">Default Label</span>
<span class="button-label primary">Default Primary</span>
<span class="button-label success">Success Label</span>
<span class="button-label info">Info Label</span>
<span class="button-label warning">Warning Label</span>
<span class="button-label danger">Danger Label</span>
```

Jumbotron

Jumbotrons can be treated as large panels, the default panel has the text entered in the centre with lots of padding all round, as seen in Figure. 14 on page 23.



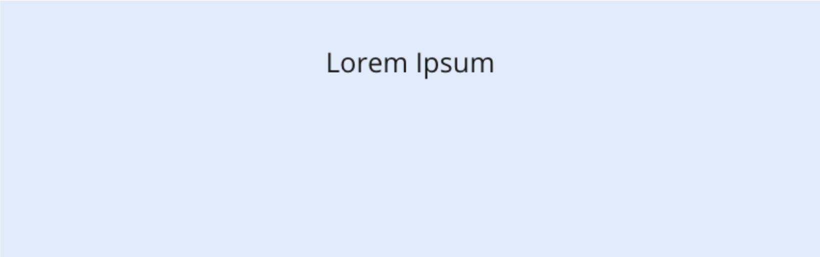
Lorem Ipsum

Fig. 13: Jumbotron

Below is the code needed to create a jumbotron:

```
<div class="jumbotron"><h1>Lorem Ipsum</h1></div>
```

As jumbotrons share similar attributes to panels, the same modifiers that apply to panels can also be applied here as seen in Figure. ?? on page ??.



Lorem Ipsum

Fig. 14: Jumbotron with Modifier

```
<div class="jumbotron alert-primary"><h1>Lorem Ipsum</h1></div>
```

Button Groups / Pagination

Button groups are a collection of buttons on the same line. Button groups work in the same way that buttons work as seen in Figure. 15 on page 24. To group a collection of buttons, add the class button-group around the collection .

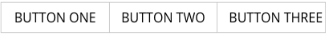


Fig. 15: Default Button Group

Below is an example of the default button group:

```
<div class="button-group"><button class="button button-default">
Button One</button>
<button class="button default-button">Button Two</button>
<button class="button default-button">Button Three</button>
</div>
```

As with buttons, the same modifiers, see Figure. 16 on page 24 can be added to either increase the button-group size.

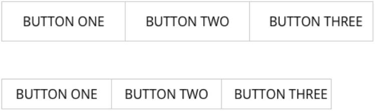


Fig. 16: Button Group Sizes

The code snippet below shows you the large button group:

```
<div class="button-group large-button-group">
  <button class="button default-button">Button One</button>
  <button class="button default-button">Button Two</button>
  <button class="button default-button">Button Three</button>
</div>
```

Or adding colour to the group see Figure. 17 on page 25.
Below is one example of modifiers:

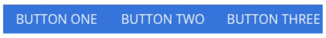


Fig. 17: Button Group With Modifiers

```
<div class="button-group">
<button class="button primary-button">Button One</button>
<button class="button primary-button">Button Two</button>
<button class="button primary-button">Button Three</button>
</div>
```

Forms

There are multiple elements that make up a form. Taylor'd UI caters for all these elements. The overall styling for the form comes from the form-styling class as seen below:

```
<div class="form-styling"></div>
```

The following inputs make the overall form see Figure. 18 on page 25. Passwords are not entered in as plain text as the type has been set to passwords, allowing for asterisks to appear instead. All form elements have placeholder text contained that when a user starts typing, the placeholder text disappears.

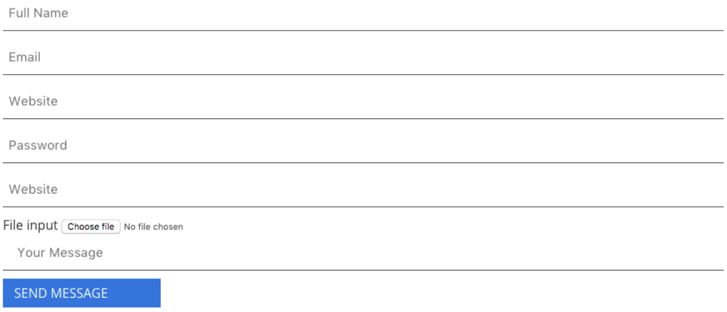


Fig. 18: Form elements

```
<input type="text" name="field1" placeholder="Full Name" />
<input type="email" name="field2" placeholder="Email" />
<input type="url" name="field3" placeholder="Website" />
<input type="password" id="password_id" placeholder="Password">
<input type="number" name="field4" placeholder="Website" />
<label for="someFile">File input</label>
<input type="file" id="fileUpload">
<textarea placeholder=" Your Message"
onkeyup="adjust_textarea(this)"></textarea>
<input class="button button-primary" value="Send Message" />
```

Other features such as radio and check boxes see figure Figure. 19 on page 26 have specific id's so that only one option can be checked at any given time.

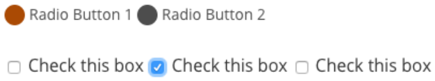

● Radio Button 1 ● Radio Button 2
□ Check this box ☑ Check this box □ Check this box

Fig. 19: Radio Buttons and Checkboxes

Below is the two types:

```
<input type="radio" id="radio01" name="radio" />
<label for="radio01"><span></span>Radio Button 1</label>
<input type="radio" id="radio02" name="radio" />
<label for="radio02"><span></span>Radio Button 2</label>
<label><input type="checkbox"> Check this box</label>
<label><input type="checkbox"> Check this box</label>
<label><input type="checkbox"> Check this box</label>
```

Navigation

The *nav* tag defines a set of navigation links. The *nav* tag is only intended for a block of high level links, this helps screen readers determine whether or not to render the content.

The navigation bar in Taylor'd UI is kept simple in keeping with the ethos of non-opinionated design as seen in Figure. 20 on page 27. The nav bar has been designed to always stay on top when a user is browsing a site on Desktop type devices. On mobile devices, the navigation bar disappears as the user scrolls a website.

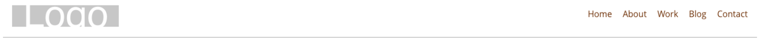


Fig. 20: Navigation Bar

In keeping with best practices, the nav should be called within a header tag as seen below:

```
<header>
  <a href="#" id="logo"></a>
  <nav>
    <a href="#" id="menu-icon"></a>
    <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>
</header>
```

960 grid, 12 column system

Taylor'd UI is based on a 960 grid system. The grid system of 12 grids. 12 columns is used as each column can be evenly divided by 960. This allows the end user to build a seamless experience from desktops to mobile devices.

The columns are also used to layout the content of a web page as seen in Figure. 21 on page 28. When using columns, it is important to remember that the columns in the section you are using them in, always match up to 12.

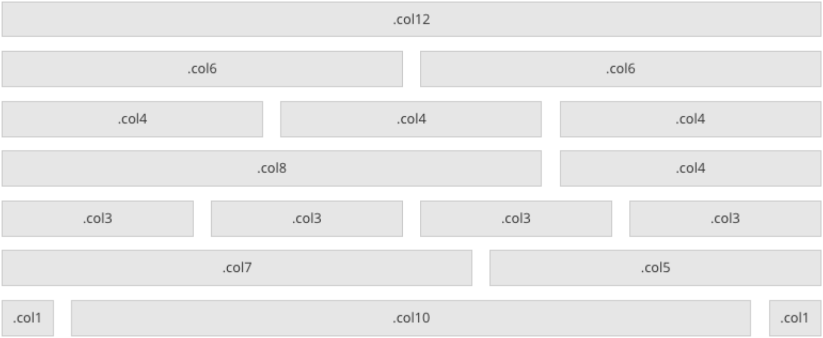


Fig. 21: 12 Column Layout

The class row-fluid is used to make the layout more fluid across devices. Media queries are also used to ensure that Taylor'd UI components collapse when they need to.

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 the website caniuse [Deveril and Schoors, 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 24. 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 template *HTML* code such as title, and links to external files such as the*CSS*. name=CSS, description=Cascading Stylesheets, sort=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 template 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.

Templates

Template's are a pre-designed webpages, or a set of *HTML* webpages that an end user can customising by adding in their own imagery, and content. The templates include all the files such as the *HTML*, *CSS*, and JavaScript files required for the templates to run smoothly.

Template's simplify the web development process, by making it easy for yyythe end user who have little or no programming experience to build their own websites.

Taylor'd UI offers three templates for the end user to utilise in their learning of *CSS*; a blog, portfolio, and product website.

While each of the template utilise the framework as their foundations. Each template also have their own *CSS* files with code that is unique to the template such as the blog template has a Read More button that expands the accompanying text section. The portfolio template has text that appears as an overlay when the user hovers over the image.

The blog template follows a simple theme, a jumbotron that has the title, and subtitle of the blog, followed by a blog post. The blog post has a title, and the date published at the top followed by an image. All the placeholder imagery is hosted by placehold.it [Nonsense, 2017].

By using an online image hosting server, the template files are smaller in size, links to the images are also less likely to break. Underneath the image, a paragraph of text is displayed, the user can click on the read more / read less button for more or less text to appear. The blog publisher and number of comments is the last section of the post.

Themes

A theme is a governed template. Unlike with templates, themes are written using *CSS*. The *CSS* manages the presentation, and graphical layout of a web page. A theme specifies the fonts, font sizes, layout of text, colour scheme, and related imagery.

Themes should be independent from the content of a website. This is so that a theme can be changed without affecting the content, causing a developer to rewrite the content.

There are currently two themes to use with Taylor'd UI, a material design theme, and a flat modern theme. The themes are built on top of the Taylor'd UI framework, and are to be used in conjunction with the framework.

The material design theme follows the google guidelines on styles [Google, 2017a]. The colours are kept to muted tones of primary colours. The font is changed to Google's Roboto for the primary font choice, and Nunito as the secondary font.

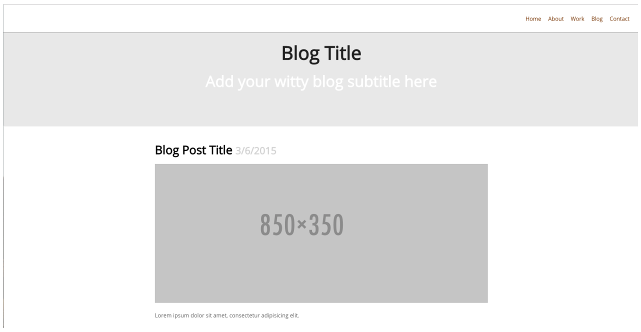
When using the material design theme, the font sizing also changes to be in line with Google's styles. In keeping with the style, a small drop shadow was also added to assets such as buttons as seen in Figure. 23 on page 35

Instead of having the generic placeholder images that are found in the templates as seen in Figure. 22 on page 35, random creative common images are taken from unsplash.com as using the unsplashit service [Marby and Yonskai, 2017].

The modern theme is similar to the material design theme, in keeping the theme understated. This theme does however has more modern, and interesting font choices such as Raleway.

A theme can be added to any of the templates by creating a link to the stylesheet as seen below.

```html
<link rel="stylesheet" href="../../assets/css/taylord.css" />
```



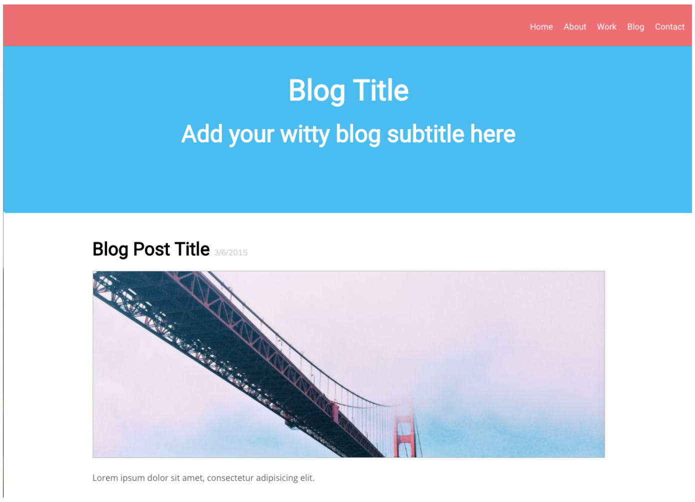
Blog Title

Add your witty blog subtitle here

Blog Post Title 3/6/2015

Lorem ipsum dolor sit amet, consectetur adipisicing elit.

Fig. 22: Generic Template



Blog Title

Add your witty blog subtitle here

Blog Post Title 3/6/2015

Lorem ipsum dolor sit amet, consectetur adipisicing elit.

Fig. 23: Template with theme

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. 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 [Project, 2017].

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 *@includes* 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.

```scss
$foundation-color: #000;
$foundation-color-2121: lighten($foundation-color,
13%);

h1, h2, h3, h4, h5, h6 {
  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.

For the neutral colours of the framework such as text and heading colours, a *SASS* colour generator was used [Arch, 2017]. The generator allowed for a base colour to be added, the output was lightened versions of the base colours. This was useful when creating the variable name, as the colour code could be entered at the end, enabling the end user to have a good idea of what the colour would look like.

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 several 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.

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

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.

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

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.

```
 <input type="submit">, <input type="image">, <input type="file">,
 <$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 25 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.

```
$breakpoint-desktop: "screen and (min-width: 48rem)
and (max-width: 60rem)";
$breakpoint-tablet: "screen and (min-width: 30rem)
and (max-width: 47.9375rem)";
$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%;
  }
}
```

Using the class row fluid, the width has changed from a set number of 960px to width of 100%. The percentage value allows for the columns to collapse in a predetermined pattern. The media query for mobile encompasses the fluid grid for it to scale down the columns.

```
.row-fluid [class^="col"] {

  @media #{$breakpoint-mobile} {
    margin-left: 0;
  }

  display: block;
  float: left;
  width: 100%;
  min-height: 30px;
  margin-left: 2.76%;
  *margin-left: 2.70%;
  @include box-sizing(border-box);
}
```

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). What is meant by keeping your code DRY, is that you are removing elements that are repeated into one section and then just calling on that section. 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;
}
```

Forms

A *HTML* form defines a form field that is used as a tool to gather user input. A *HTML* form is made up of various input elements such as checkboxes, text areas, submit buttons, password fields, and mailto links.

All types of input elements were considered, and one of each type has been added to the framework. By having one type of each input, it also allows the end user to build the element they require, using the form in the framework as their template.

All the individual element stylings are covered under the one class of form-styling. This ensures that all the elements are following the parents style.

To style the inputs, [] are used to select the type of input to style, as seen below:

```
input[type="radio"]
```

Templates, Partials and Themes

A total of three 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 templates themselves are bare, and have the look of a wireframe, as they are intended to be. The templates 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(’template/blog/pages/**/*.+(html|nunjucks)’)
  // Renders template with nunjucks
  .pipe(nunjucksRender({
      path: [’templates/blog/templates’]
    }))
  // output files in app folder
  .pipe(gulp.dest(’templates/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 %}
```

The themes themselves were kept basic due to time constraints. Currently the themes only effect the visible elements of the three templates.

When developing the themes, it was important to keep checking that the theme was not interfering with the framework. This was achieved by using Google's Inspect Element Tool. At stages through development, the link to the stylesheet was removed, to ensure that the layout from the framework was not disrupted.

In the material theme, it was important to keep the stylings similar to those set out by Google. Extra features such as drop shadows were added. These were not in the original framework as they were thought to be opinionated.

```css
.button {
  border-radius: 2px;
  font-size: 0.9em;
  background-color: #fff;
  color: #646464;
  transition: box-shadow 0.2s cubic-bezier(0.4, 0, 0.2, 1);
  transition-delay: 0.2s;
  box-shadow: 0 2px 5px 0 rgba(0, 0, 0, 0.26)
}
```

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 26. 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>
```

```

//index.html
<link rel="import" href="../iron-component-page/
iron-component-page.html">
<body>
   <iron-component-page src="example-app.html">
   </iron-component-page>
</body>
```

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.

```
//original
@media ($breakpoint-mobile)

//new method
@media #{$breakpoint-mobile}
```

Another issue in development was that the panels background were not taking the colour of the alert class specified. The classes were checked to ensure that nothing was out of place. The next step was to use Chrome Developer tools to see which backgrounds were been used. All the backgrounds were from the parent class. The code was checked again and the < was included in the class, reaffirming that the alert colour was to take precedence. As of now, the panel background colours have not been fixed.

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

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

One issue that appeared time, and time again in the framework was the the default font colour. Whenever the variable was called, the returning colour was blue, never black as the variable suggested. Changing the colour variable short code from #000 to #000000 did not solve the issue. As a work around the hex value was added to where the text needed to be black instead of using the colour variable.

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 glsHTML 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

Another feature that would be of benefit to the framework, would be the continuation of the themes. As mentioned previously the themes are quite sparse, they are only effecting the visible elements.

For future work, these themes would effect all elements that are in a generic framework.

Currently, the framework can be installed using *rem*. The goal was to build a full *CLI* that using the command line, a template and theme could be downloaded using the terminal. But due to time constraints only the *rem* was developed.

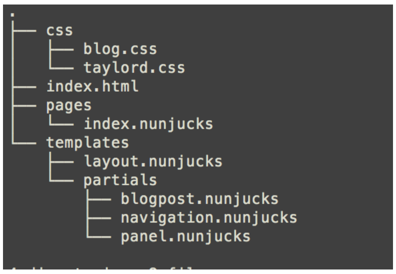
In addition to *CSS*, jQuery features to be added that would help the framework work more seamlessly such as making links active in the navigation bar, ensuring that the footer section always stays at the bottom of the page, regardless of page height, adding close options to the alerts, etc.

Conclusions

time management issues project management issues wanting to do more

Appendix

Fig. 24: Nunjucks File Structure



```
.
├── css
│   ├── blog.css
│   └── taylord.css
├── index.html
├── pages
│   └── index.nunjucks
└── templates
    ├── layout.nunjucks
    └── partials
        ├── blogpost.nunjucks
        ├── navigation.nunjucks
        └── panel.nunjucks
```

Fig. 25: 12 grid layout with the 960 grid

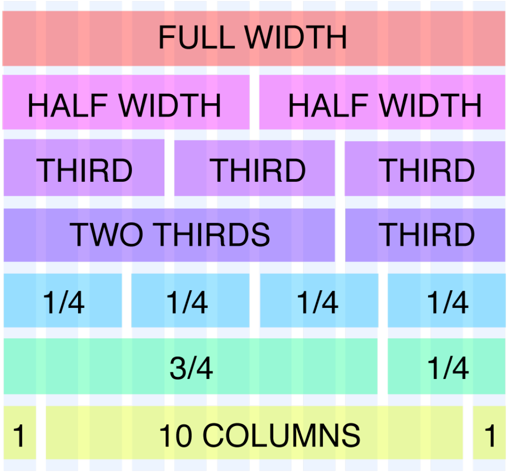
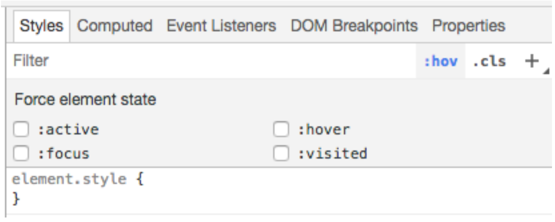


Fig. 26: Chrome Developer Tools



Force element state

□ :active □ :hover

□ :focus □ :visited

```
element.style {
}
```

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

Arch. *Sass Color Generator* , 2017. Available at: http://scg.ar-ch.org [Accessed: 21/04/2017].

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].

H. Catlin, C. Eppstein, and N. Weizenbaum. *CSS with superpowers*, 2016. Available at: http://SASS-lang.com [Accessed: 26/11/2016].

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].

A. Deveril and L. Schoors. *HTML templates* , 2017. Available at: http://caniuse.com/#feat=template [Accessed: 20/04/2017].

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].

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

Karol Król. *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].

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].

David Marby and Nijiko Yonskai. *Beautiful placeholders using images from unsplash* , 2017. Available at: https://unsplash.it [Accessed: 23/04/2017].

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].

Quality Nonsense. *A quick and simple image placeholder service.* , 2017. Available at: http://placehold.it [Accessed: 18/04/2017].

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

A. Sellier, L. Page, M. Bointon, M. Jurcovicova, M. Dean, and M. Mikhailov. *Getting Started*, 2016. Available at: http://lesscss.org [Accessed: 26/11/2016].

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 âĂŞ 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].