

Final Year Project

by Matthew Chedzey

FILE	17820_MATTHEW_CHEDZEY_FINAL_YEAR_PROJECT_78196_13769386 14.DOCX (7.69M)		
TIME SUBMITTED	21-APR-2016 02:51PM	WORD COUNT	12993
SUBMISSION ID	56207839	CHARACTER COUNT	71001



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

A Study of Mobile First CSS Frameworks

by
Matthew Chedzey

Supervisor: Kemi Ademoye
Second Marker: Michael Dacey

Project submitted as part of the requirements for the
award of BSc (Hons) Web Development

April 2016

DECLARATION

I, MATTHEW CHEDZEY declare that I am the sole author of this Project; that all references cited have been consulted; that I have conducted all work of which this is a record, and that the finished work lies within the prescribed word limits.

This work has not previously been accepted as part of any other degree submission.

Signed :

Date :

FORM OF CONSENT

I MATTHEW CHEDZEY hereby consent that my Project, submitted in candidature for the B.Sc.(Hons) Web Development degree, if successful, may be made available for inter-library loan or photocopying (subject to the law of copyright), and that the title and abstract may be made available to outside organisations.

Signed :

Date :

Abstract

This project aims to investigate the suitability of CSS3's flexbox property as a viable solution for creating responsive mobile first web pages. People now access the web on a wide variety of devices, the way we view the web is ever evolving, so should the way we create webpages. However, website layouts continue to use floats and inline-blocks along with multiple CSS 'hacks' to create even the simplest of layouts. These techniques were never intended for layouts which I why I believe it is time to embrace the flexbox property.

This report will show the results of attempting to create a CSS Framework built around the flexbox property. The results show that while not perfect flexbox is capable of being the building blocks of a website if people are willing to use it. There are still flaws and some workarounds are needed but this flaws will be fixed overtime.

Table of Contents

ABSTRACT	4
TABLE OF FIGURES	ERROR! BOOKMARK NOT DEFINED.
CHAPTER 1 – INTRODUCTION	6
AIM	8
OBJECTIVES	8
CHAPTER 2 – RESEARCH AND REVIEW OF LITERATURE	10
HISTORY OF MOBILE WEB	10
CURRENT MOBILE DESIGN METHODS	12
<i>Separate Mobile Site</i>	13
<i>Alternatives</i>	13
Native Apps	14
Hybrid Apps	15
<i>Responsive Web Design</i>	16
Graceful Degradation	17
Progressive Enhancement	18
Media Queries	19
Mobile First Design	21
Navigation	21
Frameworks	23
Flexbox	24
CHAPTER 3 –DEVELOPMENT METHODOLOGY	26
CHAPTER 4 - DESIGN & IMPLEMENTATION	27
DESIGN	27
<i>Requirements</i>	27
Navigation	29
IMPLEMENTATION	32
<i>Layout</i>	32
<i>Navigation</i>	36
CHAPTER 5 – EVALUATION	39
CHAPTER 6 – CONCLUSIONS & RECOMMENDATIONS	41
CHAPTER 7 – REFLECTION	42
CHAPTER 8 - REFERENCES	43

APPENDICES.....	46
-----------------	----

Table of Figures

FIGURE 1: HOW NUMERIC MENU SYSTEM WAS USED TO NAVIGATE SITES ON A SMALL SCREEN. (SOURCE - HTTP://LEGACY.EARLHAM.EDU/~HARRISA1/NEWECON.HTM)	11
FIGURE 2: GRAPH DEPICTING TIME SPENT USING MOBILE WEB VS DEDICATED APPS (SOURCE - FLURRY ANALYTICS:)	14
FIGURE 3: RESPONSIVE DESIGN SHOWN ON MULTIPLE DEVICE TYPES. (SOURCE - HTTP://WWW.ALABOUTTHELEADS.BLOG.COM/?p=166)	17
FIGURE 4: EXAMPLE OF MEDIA QUERIES BEING USED TO ADJUST SITE LAYOUT WHEN BROWSER WINDOW IS RESIZED.....	20
FIGURE 5 HTTP://EXISWEB.NET/MOBILE-MENU-ICONS	22

Chapter 1 – Introduction

It is no secret that mobile web browsing has grown significantly in recent times. In fact the number of global mobile users has now surpassed that of desktops and more people now spend their time accessing the internet through mobile devices rather than through traditional devices.[1] Despite these statistics, websites are still often designed with the desktop in mind first and in many instances the mobile site is a mere afterthought. This is largely down to the speed of the evolution of the mobile web experience.

When mobile phones first supported web browsing it was a very different experience to what is seen today. Sites were extremely limited due to the basic browsing capabilities of the phones, this meant that sites had to deliver a very different experience to desktop browsers. It made sense for developers to create a separate site for mobile because users were not expecting to find the same content when browsing with their phone. This practice of treating the mobile and desktop experience as two separate entities has been a habit that the web industry has struggled to free itself from. That is up until the last three or four years where designers have begun to favour a responsive web approach.

Responsive web design (RWD) is an approach to web development that allows a website to be created that can be viewed on a desktop or on a mobile device without losing any of the main functionality. RWD is not a new idea by any means but it has only become common practice in recent times. With the increased level of traffic coming from mobile sites companies started to realise that when their users visited their site on a mobile device they expected to be able to find the same content as they would on their desktop.

This created some problems as mobile devices have significantly smaller screen sizes than their desktop counterparts, because of this the layout had to be different. This would usually mean creating a single column layout and prioritising content so that the most important content is displayed first and certain non-essential elements may not be displayed at all. RWD is achieved using media queries to create break points at certain screen sizes. This would mean that a website would be designed with a desktop view in mind and then when it is viewed with a smaller screen the media queries would change the layout and remove elements. By developing it this way the site would lose some

functionality when it is viewed on a smaller device, which means that the mobile version does not have all the features that was originally intended for the site. The process of scaling down the features to a basic level so that everyone gets the basic functionality is known as graceful degradation.

According to W3.org;[2]

“graceful degradation is the practice of building your web functionality so that it provides a certain level of user experience in more modern browsers, but it will also degrade gracefully to a lower level of user experience in older browsers. This lower level is not as nice to use for your site visitors, but it does still provide them with the basic functionality that they came to your site to use; things do not break for them.”

As is evident from this definition a mobile website created using graceful degradation would give the user basic functionality although not the full experience that was decided in the original design.

The flipside to this philosophy is known as progressive enhancement. Progressive enhancement starts at a level where all users will receive the intended experience regardless of how they are viewing the site and then adding extra functionality that adds to the user experience. These two methods sound similar although with progressive enhancement, the developer can make sure that everything that they want is delivered in the basic site and any extras can be added as they scale up.

Mobile first design goes hand in hand with progressive enhancement, as the philosophy of mobile first is that you design with mobile users in mind first. This means that anything that the site needs to deliver should be designed so that it can be easily accessible to mobile users. No functionality should be removed from the site to make it work with mobiles.

Aim

The aim of this project is to develop a CSS Framework using CSS3's flexbox property.

Objectives

- Investigate the importance and relevance of mobile first design in web development.
- Research and evaluate current mobile design methods.

- Compare the impact of a web site designed using a desktop first approach to that of a mobile first designed site.
- Investigate the impact a mobile first designed site has on SEO/Google rankings.
- Test the suitability of the CSS3 property 'flexbox' for creating mobile first responsive layouts.

The structure of the report will be as follows:

Chapter 2. Literature Review

This chapter will begin with a brief history of the mobile website. This will give the reader a clear understanding of why a mobile first approach was developed. A discussion of the advantages and disadvantages of mobile first design as well as other methods that could be used.

Chapter 3. Research and Development Methodology

Chapter 3 will discuss the research and development methodologies that will be used for this project. The reasoning for choosing these methodologies will be given as well as any advantages and disadvantages.

Chapter 4. Design and Implementation

This chapter will explain how the web application will be designed and developed. The requirements of the application will be listed here as well as the reasoning for these requirements.

Chapter 5. Evaluation

This chapter will discuss the findings of this research project and compare them to the objectives that have been set out in this first chapter.

Chapter 6. Conclusions and Recommendations

Chapter six will summarise the findings of this project, any further work that could be undertaken to further expand on the topic of mobile first design will be outlined here.

Chapter 7. Reflection

This final chapter will allow the project author to discuss what they have learned both in this particular subject and personally. The author will explain what went well during the project as well as any areas that could have been improved or if a different approach could have given better results.

Chapter 2 – Research and Review of Literature

History of Mobile Web

Mobile technology has evolved through many different generations before it arrived to where we are today. According to Fling [3], there have been five distinct eras of mobile devices that have shaped the mobile devices that are used today. These are; the brick, the candy bar, the feature phone, the smart phone and the touchscreen. The “brick era” mobiles were large phones, and they were so big because they needed such big batteries to allow them to connect to the nearest network antenna, which at the time were rare and in sparse distribution. During the 90’s it became popular to have these phones in cars, this meant that more antennas were needed to provide coverage while these car phones were on the move. The fact that there was now a larger network coverage meant that the batteries could be made a lot smaller as the phones needed less power, this allowed for the creation of the second era of devices known as the “candy bar” phones.

The term candy bar has long been the term to refer to the long rectangular shape mobile device that became the standard design and is still used widely today. These phones represented a considerable leap in mobile technology. The fact that these phones could now be carried around in a person’s pocket made them much more desirable. The increased demand brought about an increased competition between not only the network providers but also the device makers, meaning the cost to consumers was reduced.

The candy bar era also introduced Short Message Service (SMS). The initial idea behind SMS was for mobile operators to send their customers notifications such as when they had received a new voicemail [3]. However, due to oversights by the mobile operators, people realized that they could send these messages to their friends for free. Thus, not only did the candy bar era open up mobile use to the masses, it also introduced a new way of communicating through text messages.

Following these developments mobiles became so popular that manufactures needed ways to make their devices stand out from the competition. A hugely popular feature that was introduced was ‘time-waster games’, the most famous of which was Nokia’s Snake. The

success of these non-communication uses of mobile phones made it clear that users had an appetite for more features on their devices.

The introduction of the Wireless Application Protocol (WAP) was the general public's first taste of using the internet on a mobile device. WAP was created in 1997 by a group known as WAP Forum, and the group consisted of mobile manufacturers Nokia, Ericsson and Motorola and Phone.com. Their initial purpose was to define an industry wide specification for developing applications over a wireless communications network [4].

Using WAP to browse the internet the same way that you do when using a PC was next to impossible. WAP used its own markup language known as Wireless Markup Language (WML). This allowed a very basic implementation of the internet on early mobile devices, its main use was for checking stock prices and sports scores, there were also sites that allowed users to download wallpapers and ringtones which allowed for personalising phones for the first time [5].

Despite this step forward for mobile web browsing, there were major drawbacks to WAP. It could not handle full web pages, this meant that completely separate websites needed to be developed, which cost extra time and money. Also due to the small nature of the screens used, it was not suited for lots of text which limited the use for a mobile site in the first place. Navigation also caused issues due to the lack of control that these devices offered, sites often made use of the handsets numeric pad to select menu items this was slow and cumbersome.



Figure 1: How numeric menu system was used to navigate sites on a small screen. (Source - <http://legacy.earlham.edu/~harris1/newecon.htm>)

Krug states that [6]:

"Mobile devices meant cramped devices, squeezing Web pages the size of a sheet of paper into a screen the size of a postage stamp."

He argues that despite numerous attempts at solving the issues of mobile browsing, it was not until the smartphone became popular that people started to take the mobile web seriously. In particular Krug credits Apple with combining the power of its handsets with the more modern mobile browsers that were capable of displaying full HTML pages on the phones touch screen enabled display [6].

Key to the iPhones success was Apple's choice to use a capacitive touch screen when most manufacturers at the time were still using resistive screens. This allowed for multi-touch support so users could make gestures like 'pinch and zoom' as well as swiping along the screen. This vastly improved navigation around a mobile website, it also meant that users could view a full desktop website on a mobile device and easily zoom in to the content that they wanted. By todays standards that would not make for very good usability but at the time it opened the mobile phone up to a truly mobile browsing experience.

The open source mobile operating system Android soon followed and the mobile web changed towards the browsing experience that users enjoy today. Along with new hardware capabilities there was also advances being made in web development practices. The evolution of HyperText Markup Language (HTML) and Cascading Style Sheets (CSS) allowed developers to create more sophisticated mobile sites that were optimised for viewing with the ever evolving smartphones.

Current Mobile Design Methods

The advancement of both hardware and software technologies opened up new ways of optimising content for mobile devices, the power of the new smartphones allowed for applications to be created that could connect to the internet and deliver content to the user while taking advantage of the devices hardware. The evolution of HTML and CSS and more advanced browsers that could take advantage of these new technologies gave developers ways of creating mobile specific sites and even ways of creating sites that dynamically changed its content based on how the user was viewing it.

Separate Mobile Site

With the introduction of more advanced web browsers, phones are now capable of displaying full websites. The problem, however, was that these websites were designed to be viewed on a desktop device. So although phones were able physically to display the website, the user was not given a great experience while browsing. This was due to the fact that they had to zoom in on the text in order to read it comfortably, meaning they had to scroll horizontally as well as vertically. A new approach to mobile design was clearly needed in order to supply the user with a good experience.

It was generally accepted by web developers that users would only use their phones for browsing the web when they needed specific information while they were on the move. This thinking led to the rise of separate mobile sites. Having a separate site meant that companies could deliver a mobile site that would be tailored to the user's needs. The site would not be bloated with information and features that was not relevant to mobile users. This did bring about some issues, by having a separate site it meant that a website would have two different URL's. This could cause issues with search engine indexing and there could be re-direct issues if someone shared a link from their mobile with their colleague who then viewed it on their desktop. The biggest issue was down to cost and maintenance, because there were two separate sites it would take twice as long to develop and could become a nightmare to maintain, especially with larger sites.

Alternatives

Due to some of the disadvantages of using a separate mobile site, developers began coming up with new ways of delivering web content to mobile users. The most popular was to create an app that could serve all the information that the users required, apps were either developed for cross-platform use or for use with specific platforms, these development methods are referred to as native and hybrid apps.

Native Apps

A popular alternative to making a mobile website is to create a native application. A native app is an app that is developed for a specific platform, e.g. iOS, Android, Windows Phone. The advantage of creating a native app is that the developer has full access to the features of the phone, for example, the camera, the accelerometer or directly accessing the phone's file system.

In 2014, Khalaf [7] conducted a study to investigate mobile usage of US consumers. Findings from the study showed that consumers spend 86% of their time on mobiles using Apps compared to just 14% using the mobile web. This is a strong indicator that people prefer the use of dedicated apps over the mobile web.

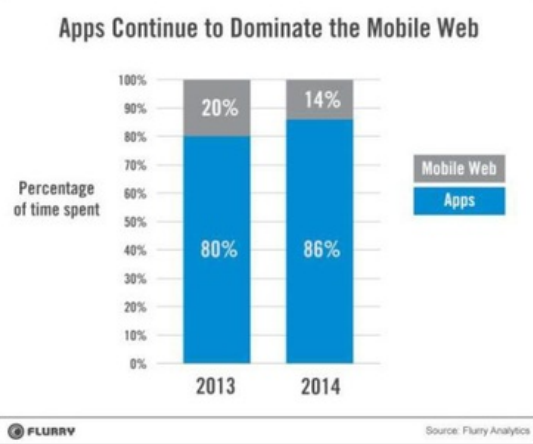


Figure 2: Graph depicting time spent using mobile web vs dedicated apps (Source - Flurry Analytics:)

Despite this strong evidence that the users prefer the use of native apps over mobile websites, there is still a strong opinion amongst developers that mobile websites offer a better option. Fling[8] argues that;

“the mobile web is the only long-term commercially viable content platform for mobile devices.”

A key reason for this view is that there is a large amount of fragmentation throughout the mobile world. Firstly, there are the different platforms such as iOS, Android and Windows Phone. If a developer wishes to cater for all users, they must create and maintain an app for each of these platforms. This is due to the fact that each platform uses a different programming language, this process is hugely time consuming and very costly. Secondly, within each of these platforms there are different versions which have different capabilities. Android is famous for having a fragmented operating system due to the fact it is available on a wide variety of devices. This means that some apps may only be available for specific platforms or only the latest versions and some of the intended users will miss out.

The mobile web on the other hand works across all devices, so developers can create one website that will work no matter how the user is viewing it. Historically the mobile web has been at a disadvantage to native apps due to it not having access to device specific features, such as the camera and GPS. However mobile browsers are evolving fast and more and more are starting to add support for these features, the more this happens the smaller the gap will be between the two approaches.

Comment [O A1]: Use more formal language

Hybrid Apps

Hybrid Apps offer some of the benefits of both native and mobile web apps, in its simplest form a hybrid app is a mobile web app that is wrapped in a native like platform specific shell. This means that a hybrid app is able to fully access the device's native features such as camera, microphone, GPS, etc but because the app itself is developed using HTML, CSS and Javascript it is compatible with all the different mobile Operating Systems.

The biggest advantage that a hybrid app has over a native app is that the development time is much faster and the finished application is much easier to maintain, this is due to the fact that native apps need to be developed separately for each mobile platform whereas a hybrid app only needs to be developed once and then wrapped in a platform specific shell to allow it to be distributed amongst the different mobile platforms.

Responsive Web Design

Despite evidence showing that people spend a lot of time using apps compared to browsing the web on a mobile device, it is still recommended that there should be a mobile presence of any website regardless of there being a native or hybrid app alternative.

Appel [9] states that;

"Regardless of whether you're going to build a native or hybrid app, you should have a mobile version of your Web site. Mobile Web sites offer the largest immediate and instant reach of all the types of apps."

This is due to the fact that mobile sites are available to everyone, regardless of which mobile platform they are using and having a mobile compatible website is said to improve a sites standing in Google search results [10].

As roaming prices came down and more people started using the mobile web more casually, it became clear that people were no longer using the mobile web for specific situations. People's browsing habits had changed, while **sat** at home people no longer went to their desks or reached for their laptops if they wanted to search for holidays or do some online shopping. Instead they would simply pull out their mobile phones and start browsing and this meant that they expected to find the same content on their phones as they did on their desktop browsers. However, due to so many sites having a mobile counterpart, this was rarely the case.

Comment [O A2]: Informal speak again – I guessing you were tired at this point 😊

The idea of RWD would bring an end to that. As HTML and CSS languages evolved, better techniques were made available to developers. The introduction of media queries allowed developers to set breakpoints that would tell the user's browser to show certain information depending on the size of the screen being used to view the site. This meant that content could be re-sized and re-flowed so that it could be made more accessible to a mobile user. *Figure 1* shows a website layout on different device sizes and orientations, you can see that the content has been re-sized and re-flowed to keep the most important and relevant information at the top of the screen.



Figure 3: Responsive Design shown on multiple device types. (Source - <http://www.allabouttheleadsblog.com/?p=166>)

Graceful Degradation

With the introduction of RWD came the challenge of creating a single website that could seamlessly adapt from a desktop view to a mobile one. One method that became very popular was that of graceful degradation. It is not just about creating web sites that can be viewed on multiple devices but also it is a way of handling different browsers. Web Browsers have been around for as long as the Web and because of this it is possible that some users may be viewing sites using old out of date browsers that are missing the functionality of the more modern browsers. Although old browsers are not the only thing that can cause problems, some people may have technologies turned off such as JavaScript, this could be down to the user using a company computer and the company does not allow JavaScript to be enabled.

According to W3.org; [2]

“graceful degradation is the practice of building your web functionality so that it provides a certain level of user experience in more modern browsers, but it will also degrade gracefully to a lower level of user in

experience in older browsers. This lower level is not as nice to use for your site visitors, but it does still provide them with the basic functionality that they came to your site to use; things do not break for them."

As is evident from this definition a mobile website created using graceful degradation would give the user basic functionality although not the full experience that was decided in the original design.

Because mobile devices have significantly smaller screen sizes than their desktop counterparts the layout had to be different. This is to ensure that the user still has a good experience while browsing. This would usually mean creating a single column layout and prioritising content so that the most important content is displayed first and certain non-essential elements may not be displayed at all. A website that is developed using graceful degradation starts with the desktop view in mind and then when it is viewed with a smaller screen the media queries would change the layout and/or remove elements. By developing it this way the developer ensures that user gets the optimal experience while viewing the site from a desktop/laptop computer. When viewed with a mobile device the site would lose some functionality, which means that the mobile version does not have all the features that were originally intended for the site.

Comment [O A3]: I am struggling to see the link here

This approach is designed with the "ideal user" in mind, that is, a user who uses the most up to date browser with JavaScript enabled. That is not to say that other users are not catered for but these users may not get the full site and its features as was intended when it was designed. Having a message saying "Please download the most up to date browser" to be displayed on the user's website does not count as graceful degradation, but would actually be classed as 'browser centric design'.

Progressive Enhancement

Progressive enhancement is a similar principle in that it promotes web sites that support all browsers but whereas graceful degradation starts with a full 'bells and whistles' site and then scales it down to support older browsers, progressive enhancement does the opposite. It starts with the lowest common denominator so that if a user views the site with an out of

date browser they will be shown the site as originally intended and then the site will be scaled up for users with more modern browsers adding more features and functionality. Gov.uk states that in web design the only part of a page you can rely on working across all browsers is the HTML as without that there is no web page and all the other stuff becomes irrelevant. They state that each extra layer, such as images, CSS, video, audio should be seen as optional.

With this in mind the first thing a developer should do is create a web page that works just using HTML, this guarantees that whatever browser the user is using they can still view the site. Once this is accomplished, the developer can then set about adding extras such as CSS and images making sure to take into account what current and old browsers are capable of displaying.

Progressive enhancement starts at a level where all users will receive the intended experience regardless of how they are viewing the site and then adding extra functionality that adds to the user experience. Although these two methods sound similar, with progressive enhancement, the developer can make sure that everything that they want is delivered in the basic site and any extras can be added as they scale up.

Media Queries

One of the key features needed to create a responsive website is the @media property. Media queries were introduced to solve the usability issues that arose when viewing a website designed for desktops on a mobile device. W3C state that [11]“A media query consists of a media type and zero or more expressions that check for the conditions of particular media features”. For example, if a site had a two column layout where the main content’s width was set to 80% and the navigation column had a width of 20%. This would look absolutely fine on a desktop monitor, however when viewed on a mobile device it would be unreadable as 20% of a mobile screen width is too small to be able to read anything comfortably [12].

Media queries are able to solve these issues by letting browsers render different styles to particular elements depending on the device the site is being viewed on. The media queries

work by checking a rule that is set by the developer and if it is true any styles that are set in the media query will be set. The media queries need to be set at the end of the style sheet, this way any rules that are set will overwrite any rules that have already been set. This allows for things like layout and content to be adjusted.

Before the introduction of CSS3 the only way to achieve anything similar was by JavaScript or through server-side detection but because media queries are run from the CSS it runs quicker and it responds in real-time, so not only does it detect the screen width when the page is loaded, the CSS will automatically adjust if the user re-sizes their browser window [13].

In the following example the default CSS value will style to container elements to be set side by side each taking 50% of the page's width, however when the browser window is reduced the containers are laid out in a single column with their width now increased to the full width of the window.

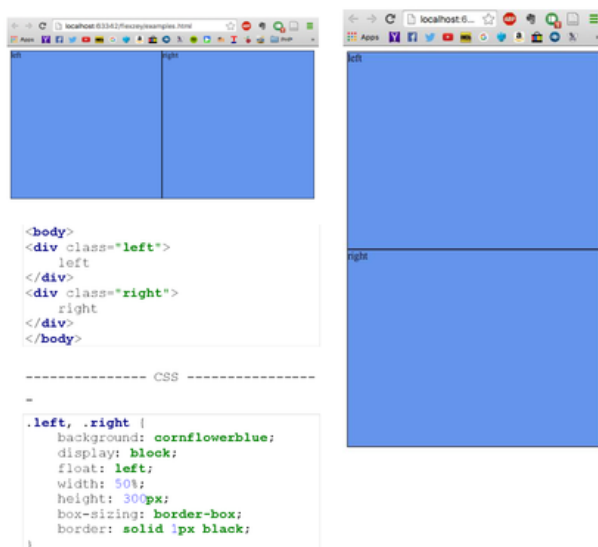


Figure 4: Example of media queries being used to adjust site layout when browser window is resized.

As well as changing the layout, media queries can also be used to alter the actual content. By setting an element's display property to none, sections can be removed from the page. This means that if a site has a sidebar that displays additional information that may not be crucial to the user's experience, that sidebar can be hidden from the page when viewed on a small device. This will save space and also make the page load quicker.

Mobile First Design

Mobile first is a design concept that requires designers and developers to cater for smaller screens first, and then to scale up to larger devices. Luke Wroblewski is often credited with defining the mobile first concept in 2009[14], [15]. Mobile first design has become an extremely popular method of web design, one of the key reasons for this is due to the massive growth in mobile usage. Wroblewski states[16]

"Mobile Web growth has outpaced desktop Web growth 8x Global mobile data traffic should grow 26x over next 5 years"

The natural constraints of mobiles force developers towards more focused, simplified solutions. The fact that mobiles have a smaller screen means that the information displayed on the screen must be relevant to the user so the important things are shown first.

There are many things to consider when designing mobile first, including, navigation and usability.

Navigation

The limited screen space available on a mobile device means that navigation needs to be approached differently when designing for mobile devices. An extremely popular method of navigation is to have an 'off-screen' menu, with this method the actual menu is hidden from view and replaced with a menu icon. This gives the user more space to view the actual website without the menu taking up valuable room on the small screen. Media queries are used to hide the menu when using a mobile device and jQuery is called when the user clicks the icon which then makes the menu visible again.

The menu icon used is often referred to as a 'hamburger menu' this is due to its shape of three lines stacked upon each other like a burger in a bun. Norm Cox is often credited with

creating the hamburger menu for the Xerox Star in 1980. Despite the fact that this icon first appeared over thirty years ago, it has only started to appear in web design in very recent times. Facebook adopted the icon in it's mobile website and app in 2010, this move by such a big company helped make the use of the icon a popular trend in web design. [17][18]

Since the introduction of the hamburger menu, many people have championed it as the solution to mobile navigation menus. This is mainly down to the fact that it solves the problem of traditional menus taking up valuable screen real estate. However, even though there is a great amount of support for this technique there is a growing amount of research that shows that it may not be the best solution. In April 2014, exisweb.net conducted two sets of AB tests to test how well the hamburger menu performs. Three different variations of the hamburger icon were used to determine how well users understand and interact with the icon. Figure 2 shows the results of these tests, the conversion rate shows how many users actually clicked the icon.

VARIATIONS	VISITORS	CONVERSIONS	CONVERSION RATE	IMPROVEMENT	CHANCE TO BEAT BASELINE
Navicon plus border	6,512	157	2.4% (±0.37%)	+22.4%	95.8%
Navicon plus MENU	6,579	139	2.1% (±0.35%)	+7.2%	71.9%
Original BASELINE	6,649	131	2.0% (±0.33%)	---	---

Figure 5 <http://exisweb.net/mobile-menu-icons>

The results clearly show that although there is very little difference between the three icon types, only 2% of users actually clicked the icon. This is seen as a problem as it means a big portion of the site is unavailable to the users due to them not knowing how to access the menu. It can be argued that as the icon is still relatively new to the world of mobile web design, that users just are not used to it yet. However, many have countered that argument by stating that the reason users do not know what the icon means is not because it is a new icon but because the icon itself is not intuitive. Rand-Hendriksen states that;[19]

"The problem of course is that it doesn't look like a menu unless you already know that it's supposed to look like a menu. What we have here is

a classic case of theory dependence and tacit knowledge: The designer and everyone involved in the application being built know from experience that the hamburger means a menu and thus they assume it's intuitive."

There are other factors to take into account when designing a navigation menu for mobile sites, such as its placement. As mobile phone screen sizes are getting larger it is becoming increasingly difficult for users to comfortably reach the top of the screen, which is where the traditional navigation menu is placed. Wroblewski argues that because mobile devices are rapidly changing and more devices are becoming touch-enabled, web navigation should no longer be split between desktop and mobile styles but instead navigation should be placed along the bottom with large touch targets [20].

Although having the navigation along the bottom of a page is intuitive for a mobile device due to the user's hand placement, this is not the case for desktop users. Despite more laptops and even some desktops having touch screen displays there is little evidence to suggest that users make use of this feature to navigate the web. Also there is still lots of evidence to show that users spend most of their time looking at the top of the page as that is where the content that they are searching for is likely to be [21]. If navigation was moved to the bottom, then users might overlook it the same way they currently overlook hamburger menus.

Frameworks

CSS frameworks are a set of pre-written style sheets that can help speed up creating websites that are responsive and cross-browser compliant. Frameworks usually contain some standard parts such as a style reset, as browsers can behave differently when rendering the same elements, it is common for developers to set a number of style resets at the beginning of their style sheet [22].

This will typically involve removing borders, padding and margin from elements as well as setting a default font and size, this will ensure the website will always look the same no matter what browser the user is viewing it with. Because a reset will need to be performed each time a developer creates a new site, it can become tedious and time consuming to do

this each time. Making use of a framework eliminates this task as the framework file already contains all these reset styles, all the developer needs to do is add the framework's CSS file to their project.

Other sections that a framework might include are typography, this is a set of rules that establishes properties such as font families, line heights, etc. If the framework is responsive, then any size related CSS will be defined as percentages or ems which will relate to the default font size.

The main section of a CSS framework will include numerous classes and ids that can be set to elements to allow for easy and quick implementation of common features. For example by creating an unordered list inside a nav element with the appropriate class will automatically style a navigation menu without the developer having to spend time adjusting the CSS properties.

A lot of CSS frameworks will combine these different sections into one CSS file to allow a developer to cover any design that they wish to create. Others will allow developers to just take the part that they need, this way if the developer is just after something that will help them create a responsive grid system they only need to use that specific CSS file. By splitting the files, the developer has more flexibility and is able to keep the overall size of the website down by not including a massive CSS file, most of which the developer might not even use.

Flexbox

Flexbox is a CSS property that allows content to be laid out responsively in a website, it's full name is Flexible Box Layout Module. According to W3C flexbox was intended for positioning elements such as navigation menus rather than overall page layout [23]. This is because another property called Grid Layout Module has also been developed alongside it and that was developed specifically for page layouts. Unlike the flexbox model, grid layout has very little browser support at the moment, so it has largely been overlooked in favour of flexbox.

The key difference of flexbox compared to other layout techniques such as using floats, is that it has been developed with responsive design in mind. Elements inside a flexbox

enabled container can adapt to the available space and even automatically wrap, this means the layout will not break when the browser window size is changed.

Despite the seemingly vast improvement over current layout techniques there are still questions over flexbox's suitability as a layout method [24], although a lot of this can be attributed to poor browser support. However, browser support for flexbox is increasing fast and at the time of writing this all the latest versions of the major browser fully support flexbox, with the exception of Internet Explorer.

Chapter 3 –Development Methodology

Chapter 3 will discuss the research and development methodologies that will be used for this project. The reasoning for choosing these methodologies will be given as well as any advantages and disadvantages.

During this project the prototyping methodology will be used. Prototyping is a common methodology in software development, this is because it allows for iterative testing from the beginning and throughout the development cycle. Because of this, errors can be picked up early on and fixed, also while developing new ideas maybe uncovered which can then be added to the design requirements. Also it could be discovered while testing that an idea may not actually be suitable and therefore can either be adapted at that time or removed from the requirements entirely.

Ralhan states that[25] there are three types of prototypes; throwaway, evolutionary and incremental. For this project the evolutionary prototype would be best as it suits the nature of how best to develop an experimental framework. Evolutionary prototypes start by developing an initial software/application and then testing it, the part that is tested can then be either improved or signed off.

A disadvantage of this methodology is that the project could easily grow beyond its original scope. This is because there are no set design requirements at the beginning and through each iteration new and improved features are constantly added. To overcome this issue, it has been decided to have a requirements catalogue to state what the finalised framework should be able to provide.

Chapter 4 - Design & Implementation

This chapter will discuss the choices made when designing the CSS framework and the reasons behind those choices. A requirements list will be produced stating what will be created, this can then be used to refer to when evaluating the final outcome. Sketches that have been used to come up with a design for any component will be shown here along with justifications for why these designs were chosen.

There will also be an implementation section will discuss how they framework was created throughout the development cycle. Testing carried out will be discussed here along with the outcome of those tests and any changes that need to be made.

Design

While designing a new CSS framework that will help developers create websites that follow mobile first principles, a list of requirements is needed that explain which features that relate to mobile first development need to be catered for. By creating a requirements catalogue it will be easy to test the framework to make sure it meets the intended criteria. This will be useful throughout the development of the framework, as it will make sure that no features are missed during the development cycle and also while evaluating the final outcome.

Requirements

- Responsive Grid

A key requirement for any mobile friendly website is that it can adapt to however the user may be viewing it. By making use of a responsive grid layout, in particular the 'flex-box' property, it should make it very simple to create responsive grid layouts from scratch.

- Base layer

Different browsers render CSS elements in different ways it is important to create a set of base styles that normalise the CSS so that all browser act in a uniformed way.

- Navigation menus

All websites need some form of navigation; therefore, it is important that this framework can easily provide different types of navigation menus. It will be

important that these menus work on mobile devices as well as desktops. The user should be able to create a horizontal or vertical menu with ease.

- Forms

Creating forms that allow for easy input from all device types is an important requirement for any framework. There should be options for the user to easily change the style of the input boxes to match the style of their site, i.e. rounded corners, flat design, etc.

- Buttons

The framework should also give the user the ability to easily change the look of any buttons that they might have, including having an option to give it an the appearance of the button being selected.

- Cross-Browser support

The framework will need to be tested across different browsers, both desktop and mobile variants.

- Media

There should be the ability to add images and videos to a web page, this media box will need to be fully responsive so it can adapt to any page size. The user will also have the option of adding a border and caption to their media. The framework will also have classes that will allow the user to easily add a image gallery to their site.

- Typography

There will be a typography module that the user can add any fonts that they wish to, they will also be able to change any default values in here to their own preferences, such the font size and line height values.

For a CSS Framework to be easily usable it must make use of a simple naming system that its users can easily identify with. The decision has been taken that as the grid will be made up of rows and columns that it makes semantic sense to use this in the naming of the grid layout. Therefore, each row of the grid will simply need a class named 'row' and for each column in that row the class will simply be 'col', by using the flex display property the user can just add multiple div elements with the class 'col' and the flex property will automatically divide the columns equally.

There will be cases where the user will want to specify how much space an element will take up on a page and because of this the user will also be able to specify this by slightly altering the 'col' class. The grid will be based off of a 12 column system, this is popular amongst CSS grid systems as it offers great flexibility as it is divisible by 1,2,3,4 and 6, allowing for a variety of different combinations while still keeping evenly distributed columns of equal size.

So to keep the naming system simple and easy to remember, the user simply needs to add a hyphen and a number that indicates the number of columns that the element needs to take up.

```
<div class="row"> // class set to row to signify a new row being started
  <div class="col-6">half</div> // class set to col-6 to set the column to take up half the
  row
  <div class="col">quarter</div> // class set to default 'col' so the remaining columns are
  divided equally between the remaining space
```

Navigation

Navigation is an extremely important aspect of any website, especially when it comes to the mobile version of a site. This framework will offer two different navigation options both of which follow mobile first principles.

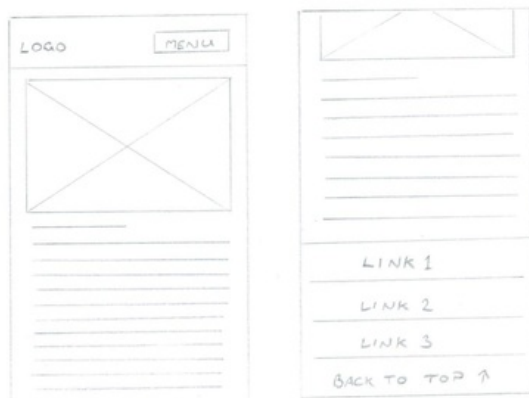
When designing a navigation for mobile use it is important to remember that content should take precedent over navigation. This is because the way users interact with a mobile device differently to how they might with a desktop browser. They are likely to be on the move and are will be looking for very specific information, because of this they will more than likely arrive at the page they want via a search engine or maybe a link that has been shared with them. As they are already at the correct page they should expect to be greeted by the sites content first and the navigation should be secondary, although still easily accessible.

If the navigation is kept at the top of the page, as it is on desktop sites, it could often obscure the content of the site when it is loaded forcing the user to scroll past the menu to reach the content they are after. To overcome this issue, it has become common practice to

hide the navigation when the site is accessed with a mobile device and allow the user to access it with a button. This button is often signified with a 'hamburger' icon but as was covered in chapter two of this report, the hamburger icon can often cause confusion amongst users. Because of this the decision has been taken to simply use the word 'menu' to signify to the user that they can reach the navigation menu by pressing this button. To further help the user it has been decided that the button should have a large touch target and be given good affordance, by making it obvious that it is intended to be pressed. This can be achieved with the use of borders and a contrasting colour, making it more intuitive to the user.

The first of the two navigation designs will place the navigation at the bottom of the page with a simple button that will take the user to the menu if they need it. Having the navigation links at the bottom of the page will mean that they will not interfere with the page content when the page loads and jQuery is not needed to show the links to the user.

Media queries will be used to move the navigation links to the top of the page when viewed on larger screens. Some older browsers do not support media queries, by designing the menu like this it will mean that even users that still use these browsers will simply be presented the mobile navigation which although not optimised for desktops is still



functional and easy to use.

To save the user having to scroll all the way back to the top after looking at the menu there will be a link that will take the user back to the top of the page, this link will be removed from the navigation when the site is displayed on a larger screen.

The second design will use a more familiar 'drop-down' approach, where the user will click the menu button and the navigation links will appear by dropping down from the header and appearing over the top of the site's content. jQuery will need to be used for this approach so the browser knows when the user has clicked the button. To aid with usability the menu button will change to a close button so the user knows that they are able to close the menu by clicking the button again.



If the site is viewed on a larger device, then a standard horizontal menu will be displayed.

Implementation

Layout

The first step to creating this CSS Framework was to develop a layout grid system, this would allow users of the framework to be able to quickly create simple layouts for any sites that they could use numerous times.

The flex box model was being used to create the layout, this meant that by simply creating two classes called `.row` and `.col` and setting the appropriate properties a very basic grid system could be created that would automatically adjust depending on the device size used to view the page.

```
.row {  
  display: flex;  
  flex-wrap: wrap;  
  align-items: flex-start;  
}  
  
.col {  
  padding: 5px;  
  background: cornflowerblue;  
  text-align: center;  
}
```



As the screenshots show the grid automatically adjusts based on the screen width, the use of the 'flex-start' property allows the heights of each div to adjust independently without affecting the flow of the row and by setting the row to wrap it allows the div containers to continue on a new row when the screen width gets too small.

A number of classes were created that allowed the user to adjust certain elements of the layout for example in the example above each column is aligned to the top of the the parent container, the classes `.alignEnd` `.alignStretch` and `alignCentre` will change how the columns will be displayed in relation to their containing row.

When laying out a web page it is common to have a gap between divs to make the page easier to read. To add a gap the class `.rowGutter` has been created. When testing this it was



discovered that when a border was added to the columns they would no longer be divided equally along the row.



This was because by default the borders and padding of containing elements were not taken into account when the browser calculates how much each space the containers took up. To fix this issue the CSS3 box-sizing property needed to be applied to any elements that could be used in the layout of the pages. The result was that the browsers included the padding and borders when calculating the width of any elements.

While it was now possible to create a simple layout using the classes created, there was no way of creating more specific layouts, for example a site may require a three column layout consisting of a left and right column taking up 10% of the page width each and a central column that takes up 80%. This would require classes that would allow a user to specify how big each column would need to be.

As was specified in the design requirements this framework needs to be able to produce a 12 column grid system, this meant that there needed to be twelve separate classes that specify twelve different column sizes. To work out the size of each column variant a simple

calculation was made. The flexbox property has an attribute called 'flex-basis' and this defines the default size of an element, it accepts multiple values including percentages. By setting the flex-basis as a percentage all that needs to be worked out is what percentage each size will take up and to do this the following calculation was made;

$$100/C = X\%$$

100 refers to 100% of the width and C is the number of columns that can fit along the rows, from 1 to 12. For example to calculate the width of a single column, C = 12 because a row is made up of 12 single columns. To calculate the width of a column that spans 3 columns then C = 4 because 3 goes into 12 4 times.

$$1 \text{ column} = 100/12 = 8.3\%$$

$$2 \text{ columns} = 100/6 = 16.6\%$$

$$3 \text{ columns} = 100/4 = 25\%$$

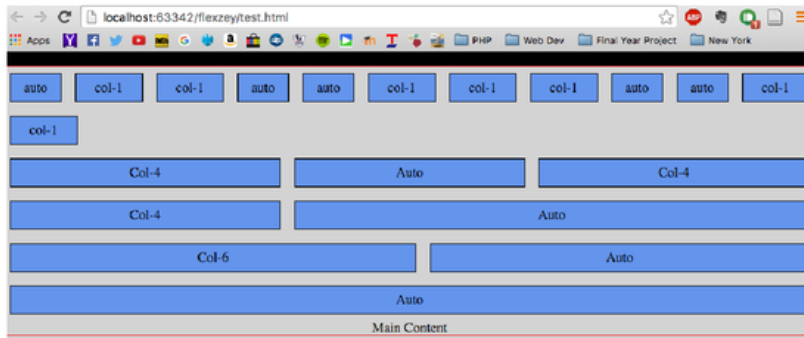
... etc.

To test that the grid worked correctly a 12 column grid layout was created using a mixture of containers with auto and specified sizes.

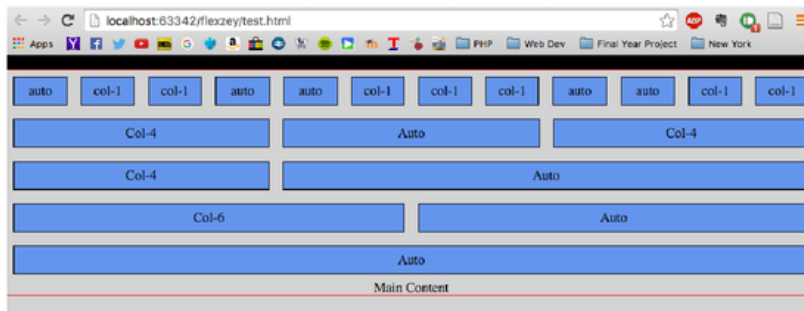


auto	col-1	col-1	auto	auto	col-1	col-1	col-1	auto	auto	col-1	col-1
Col-4				Auto				Col-4			
Col-4				Auto							
Col-6						Auto					
Auto											

The results across different browsers show the grid successfully dividing each element equally and by the specified amount. The grid was also tested using the `.rowGutter` class. This broke the layout as it no longer displayed each column on a single row, the reason for this is because the `.rowGutter` class adds a 0.5em margin to each column which is not included in the width of each column.



Fortunately the flex-basis property also accepts a calculation using the 'calc()' attribute, so to fix this issue the margin value simply needed to be removed from the width percentage like this; `calc(8.3% - 1em)` 1em is used because there is a 0.5em margin on the left and right of the column.



Caniuse.com shows browser support for the 'calc()' attribute in all modern browsers[26]. It does show that Opera Mini does not support it however testing showed this to be incorrect.

Browser support for the 'calc()' attribute (values are in percentage units, i.e. width: calc(100% - 3em))

Browser	Edge	Firefox	Chrome	Safari	Opera	Go Safari	Opera Mini	Android	Chrome for Android
Current aligned	47	47	47	47	47	47	47	47	47
Edge	9	9	9	9	9	9	9	9	9
Firefox	11	13	45	50	9.1	36	9.2	8	47
Chrome	14	46	51	TP	37	9.3			
Safari	47	52	54						
Opera	40	53							

Despite calc() being supported by all browsers there is a bug within IE where it does not work when used to calculate the width of a flex object. During testing the calc() attribute did not fix the issue of adding the margin when using Internet Explorer.

A work around was created by setting the flex-basis as auto and then explicitly setting the width of the container using the calc() attributed, this solved the issue.



```
.rowGutter .col-1 { flex: 0 0 auto;
width: calc(8.3% - 1em); }
.rowGutter .col-2 { flex: 0 0 auto;
width: calc(16.6% - 1em); }
```

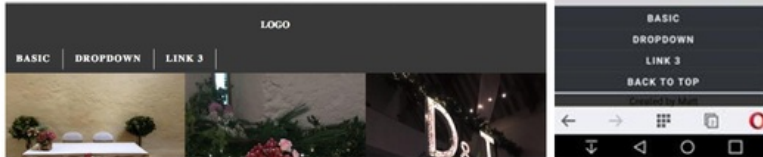
As this framework is mobile first the default styles are for the mobile layout, this way when the site is viewed on a mobile device all the style rules that are set in media queries for larger screen is not loaded by the browser. This will make the page smaller meaning it will download quicker, which is important for mobile users as they may have download limits and a slower internet connection.

Navigation

After the ability to create a responsive layout was achieved, the next step was to add navigation options to the framework. As was stated in the design section of this report, there should be two options for navigation. The first option places the navigation links at the bottom of the page, with a button in the header that takes the user to the links. This method was achieved by simply adding a link at the top of the page which acts as the button. When the user clicks the link the page scrolls to the bottom where the user can either select a link, or if they wish to go back to the top of the page, they can click the 'Back to Top' link.

When viewed on a larger screen the links are changed to a horizontal view so they can take advantage of the extra space on a larger screen. As the links are originally set along the bottom of the page they needed to be moved to the top, as is traditional with desktop websites. Usually to move elements on a page jQuery would be needed, however, because the page has been laid out using flexbox I could simply change the order of the elements. The nav footer is moved from the bottom of the page to directly under the header.

```
.mainheader {order: 1;}
.navfooter { order: 2;}
.mainContentWrapper { order: 3;}
.mainfooter { order: 4;}
```



The second navigation option uses the same button in the top right corner but instead of using a link to a menu at the bottom of the page, the menu is hidden when viewed with a mobile device. The menu is displayed when the user clicks the button, this is achieved using jQuery.

This

```
/**
 * Created by matchedzey on 11/04/2016.
 */
$(document).ready(function() {
  //Functions go here
  $("#navToggle a").click(function(e) {
    e.preventDefault();

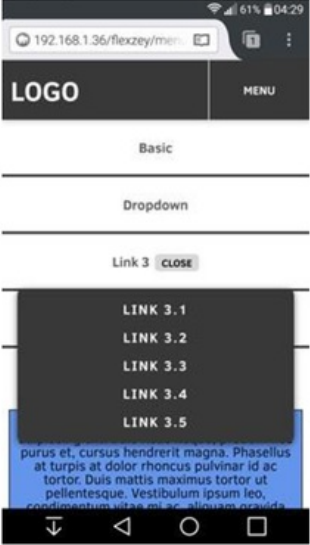
    $(".header > .dropdown").slideToggle();
    $("#logo").toggleClass("menuUp menuDown");
  });

  $(".header > .dropdown > ul > li > a").click(function(e) {
    if($(this).siblings().size() > 0 ) {
      e.preventDefault();
      $(this).siblings().slideToggle("fast")
    }

    $(this).children(".toggle").html($(this).children(".toggle").html() == 'close' ?
    'expand' : 'close');
  });
});
```

This jQuery code performs two functions, the first toggles the classes .menuUp and .menuDown when the menu button is clicked, this simply either shows or hides the dropdown menu. The second part controls any sub menus that might exist, it first checks to see if a link has any sublinks. If it does then the word 'expand' is placed alongside the link, if that link is then expanded the text changes to close. This lets the user know that there are

submenus, this is important because mobile devices can not take advantage of the hover property.



Chapter 5 – Evaluation

This chapter will discuss the findings of this research project and compare them to the objectives that have been set out in the first chapter.

One of the objectives set out in the introduction was to test the suitability of the CSS3 property flexbox for creating mobile first websites. During the development and testing of the Flexzey framework a few issues were discovered, these were mainly due to the different ways in which some browsers implement certain flexbox properties.

For example, as discussed in the Implementation section in Chapter 4 of this report, the `calc()` attribute is supported by all the browsers however Internet Explorer (IE) ignores it if it is used to calculate the size of an element with the flex-basis property. A work around was found; however, this shows the challenges in using a new technique. I believe however that when compared to other techniques that currently exist for creating a websites layout, the flexbox technique is a far better alternative. The current techniques that are used, such as *inline-block* and *float*, also have their own issues that require workarounds. The big advantage that these methods do have over flexbox is that their support amongst older browsers is far greater.

Flexbox has great support on all the modern browsers with the exception of Internet explorer, which only has partial support. This is down to the large numbers of bugs present, such as the one mentioned earlier, these bugs however can largely be overcome with simple workarounds. The larger issue comes with older versions of IE which do not offer any support for flexbox, these browsers would require a fall back option to allow these older browsers to still display a website that is designed with flexbox.

The question would be for the developer to decide, do they want/need to support these older browsers? The answer for this would usually be yes as it is considered best practise to try and cater for all users, not everybody has the option of upgrading/changing their browser and these people should not be excluded from a website because a developer has not supported their browser.

The research carried out during the literature review suggested that flexbox was not suitable for webpage layouts, however, during the testing and development of the Flexzey framework I would have to conclude that this is no longer the case. There is enough support throughout all the major browsers to use this technique alongside a fall back option for older IE browsers.

The overall aim of this project was to create a CSS Framework using CSS3's flexbox property. This aim was partially met, a framework was created that allows a developer to layout a website quickly and easily. However, the framework could be extended to include more features, such as image carousels and lightboxes, contact forms and other features that are commonly used.

These features were originally planned, however, due to the issues with getting a uniformed layout to work across all browsers and other issues discussed in the implementation section. There was simply not enough time to implement these features to a high enough standard. Despite this the framework is usable in its current form and is helpful in creating a mobile first and fully responsive website and the more advanced features can easily be added at a later date.

Chapter 6 – Conclusions & Recommendations

If I was to do this project again I would start the development of the framework much earlier, this would give me time to fix any issues encountered and find solutions to browser incompatibilities.

I would also get other developers to use the framework so I could get feedback from people who would be using it. This would help me find any issues much easier, especially and usability issues. As I have created the framework, I know how to use it therefore I might notice and flaws in the way it is designed. By having others use it, I could see how developers in the real world interpret the way I have structured my classes.

I believe that making use of CSS pre-processors for a project would have been beneficial, I chose not to go down this route due to not wanting to add more pressure by having to learn another new technique. Although looking back not only would have it made the use of the framework easier but it would have given me a new skill, so if I had a chance to do this project again I would definitely attempt to use one in the development of my framework.

Chapter 7 – Reflection

I am pleased with how well flexbox performed at creating a responsive layout, however I feel the framework is underdeveloped in comparison to what I wanted it to be. There are many features that could have been added as was discussed in the Evaluation chapter of this report. The reasons why I was unable to complete more of my framework was mainly because I did not account for the issues that I would encounter trying to get a layout to work the same on all browsers and different screen sizes. I planned my time on the assumption that very little would go wrong.

I believe that I have learned a lot throughout this project, particularly in terms of my problem solving skills and critical thinking. It took me a while to grasp the complexities of how a responsive framework is made and this made me appreciate how different CSS classes interact with each other.

I found that I was not very good at recording my testing as I went along, this caused me problems when I came to write it up as I had to rely on memory for which tests I had performed and what the outcome was. This caused real issues if I wanted a screenshot of the problem/solution as I meant I had to try and recreate the fault which was rather tedious after I had spent so long trying to fix it. This experience has made me more aware of how my current actions could affect other things down the line in a work environment.

Overall I believe I have performed well during this project despite there being aspects that I would do differently. Hopefully I can learn from this experience and improve during my next project.

Chapter 8 - References

- [1] D. Bosomworth, "Mobile Marketing Statistics 2015," *Smart Insight*, 2015. [Online]. Available: <http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/>. [Accessed: 14-Oct-2015].
- [2] W3C Wiki, "Graceful degradation versus progressive enhancement," 2012. [Online]. Available: http://www.w3.org/wiki/Graceful_degradation_versus_progressive_enhancement. [Accessed: 15-Oct-2015].
- [3] B. Fling, *Mobile Design and Development: Practical Concepts and Techniques for Creating Mobile Sites and Web Apps*. O'Reilly, 2009.
- [4] S. P. Savino, "WAP: wireless application protocol-wireless wave of the future," *Manag. Eng. Technol. 2001. PICMET '01. Portl. Int. Conf.*, vol. 1, pp. 178–179 vol.1, 2001.
- [5] J. F. Clark, "History of Mobile Apps.pdf." [Online]. Available: [http://www.uky.edu/~jclark/mas490apps/History of Mobile Apps.pdf](http://www.uky.edu/~jclark/mas490apps/History%20of%20Mobile%20Apps.pdf). [Accessed: 13-Apr-2016].
- [6] S. Krug, "Mobile: It's not just a city in Alabama anymore," in *Don't Make Me Think - Revisited: A Common Sense Approach to web usability New ...*, 2000, pp. 143 – 163.
- [7] S. Khalaf, "Apps Solidify Leadership Six Years into the Mobile Revolution," *Flurry*, 2014. [Online]. Available: <http://flurrymobile.tumblr.com/post/115191864580/apps-solidify-leadership-six-years-into-the-mobile>. [Accessed: 05-Nov-2015].
- [8] B. Fling, "Mobile Web Apps Versus Native Applications," in *Mobile Design and Development: Practical Concepts and Techniques for Creating Mobile Sites and Web Apps*, O'Reilly, 2009, pp. 143–151.
- [9] R. Appel and Microsoft, "Modern Apps : Mobile Web Sites vs. Native Apps vs. Hybrid Apps," *Msdn Magazine*, 2014. [Online]. Available: <https://msdn.microsoft.com/en-us/magazine/dn818502.aspx>. [Accessed: 19-Apr-2016].
- [10] "Mobile SEO Overview | Mobile Friendly Websites | Google Developers." [Online]. Available: <https://developers.google.com/webmasters/mobile-sites/mobile-seo/>. [Accessed: 19-Apr-2016].
- [11] "Media Queries." [Online]. Available: <https://www.w3.org/TR/css3-mediaqueries/>.

- [Accessed: 19-Apr-2016].
- [12] B. S. Gardner, "The spark of innovation begins with collaboration," *Insid. Digit. Ecosyst.*, vol. 11, no. 1, pp. 13 – 19, 2011.
 - [13] J. Bryant and M. Jones, *Pro HTML5 Performance*. Berkeley, CA: Apress, 2012.
 - [14] B. Frost, "Mobile-First Responsive Web Design | Brad Frost," 2011. [Online]. Available: <http://bradfrost.com/blog/web/mobile-first-responsive-web-design/>. [Accessed: 24-Nov-2015].
 - [15] L. Wroblewski, "Mobile first: Luke Wroblewski interview | Webdesigner Depot," 2013. [Online]. Available: <http://www.webdesignerdepot.com/2013/04/mobile-first-luke-wroblewski-interview/>. [Accessed: 24-Nov-2015].
 - [16] L. Wroblewski, "Mobile First," *Mob. First*, no. October, p. 112, 2011.
 - [17] "The Story Behind the Infamous Hamburger Icon | activemobi.com." [Online]. Available: <http://activemobi.com/blog/the-story-behind-the-infamous-hamburger-icon>. [Accessed: 03-Dec-2015].
 - [18] Antonio, "A Brief History of the Hamburger Icon | Placeit Blog." [Online]. Available: <https://blog.placeit.net/history-of-the-hamburger-icon/>. [Accessed: 03-Dec-2015].
 - [19] M. Rand-Hendriksen and E. Perakakis, "The Hamburger is Bad for You - mor10.com," 2014. [Online]. Available: <http://mor10.com/hamburger-bad/>. [Accessed: 03-Dec-2015].
 - [20] L. Wroblewski, "Responsive Navigation: Optimizing for Touch Across Devices," 2012. [Online]. Available: <http://www.lukew.com/ff/entry.asp?1649>. [Accessed: 19-Apr-2016].
 - [21] "19 Things We Can Learn From Numerous Heatmap Tests." [Online]. Available: <http://conversionxl.com/19-things-we-can-learn-from-numerous-heatmap-tests/>. [Accessed: 19-Apr-2016].
 - [22] J. Snell, "FLEXIBLE EVERYTHING Getting Responsive With Web Design."
 - [23] W. C. C. Recommendation, "CSS Flexible Box Layout Module," 2012. [Online]. Available: <https://www.w3.org/TR/css-flexbox-1/>. [Accessed: 21-Apr-2016].
 - [24] "Don't use flexbox for overall page layout - JakeArchibald.com." [Online]. Available: <https://jakearchibald.com/2014/dont-use-flexbox-for-page-layout/>. [Accessed: 21-Apr-2016].
 - [25] P. Ralhan, "prototyping." .

- [26] "Can I use... Support tables for HTML5, CSS3, etc." [Online]. Available: <http://caniuse.com/#search=calc>. [Accessed: 20-Apr-2016].

Appendices

CSS for grid layout, these styles create the responsive grid.

```
body {  
}  
  
* {  
  box-sizing: border-box;  
}  
  
#wrapper {  
  display: flex;  
  flex-direction: column;  
  margin: 0 auto;  
  min-height: 100vh;  
  background: lightgrey;  
}  
  
mainContentWrapper {  
  flex: 1 0 auto;  
}  
mainContent {  
  /*padding-top: 300px;*/  
  max-width: 960px;  
  margin: 0 auto;  
}  
  
mainfooter, mainheader {  
  background: #383838  
}  
  
img {  
  width: 100%;  
}  
row {  
  display: flex;  
  flex-wrap: wrap;  
  align-items: flex-start;  
}  
  
rowGutter > div {  
  margin: 0.5em;  
}  
  
col, col-1, col-2, col-3, col-4, col-5, col-6,  
col-7, col-8, col-9, col-10, col-11, col-12 {  
  flex: 0 0 100%;  
  padding: 0.5em;  
  background: cornflowerblue;  
  text-align: center;  
  border: solid 1px black;  
}  
rowGutter col, col-1, col-2, col-3, col-4, col-5, col-6,  
col-7, col-8, col-9, col-10, col-11, col-12 {  
  flex: 0 0 auto;  
  width: calc(100% - 1em);  
}  
  
/*-----XSMALL SCREEN DEVICES -----*/  
  
@media only screen and (max-width: 26em) {  
  col { flex: 1 0 auto; }  
  col-1 { flex: 0 0 8.3%; }  
}
```

```

.col-2 { flex: 0 0 16.6%; }
.col-3 { flex: 0 0 25%; }
.col-4 { flex: 0 0 33.3%; }
.col-5 { flex: 0 0 41.6%; }
.col-6 { flex: 0 0 50%; }
.col-7 { flex: 0 0 58.3%; }
.col-8 { flex: 0 0 66.6%; }
.col-9 { flex: 0 0 75%; }
.col-10 { flex: 0 0 83.3%; }
.col-11 { flex: 0 0 91.6%; }
.col-12 { flex: 0 0 100%; }

rowGutter_col { flex: 1 0 auto; }
rowGutter_col-1 { flex: 0 0 auto;
width: calc(8.3% - 1em); }
rowGutter_col-2 { flex: 0 0 auto;
width: calc(16.6% - 1em); }
rowGutter_col-3 { flex: 0 0 auto;
width: calc(25% - 1em); }
rowGutter_col-4 { flex: 0 0 auto;
width: calc(33.3% - 1em); }
rowGutter_col-5 { flex: 0 0 auto;
width: calc(41.6% - 1em); }
rowGutter_col-6 { flex: 0 0 auto;
width: calc(50% - 1em); }
rowGutter_col-7 { flex: 0 0 auto;
width: calc(58.3% - 1em); }
rowGutter_col-8 { flex: 0 0 auto;
width: calc(66.6% - 1em); }
rowGutter_col-9 { flex: 0 0 auto;
width: calc(75% - 1em); }
rowGutter_col-10 { flex: 0 0 auto;
width: calc(83.3% - 1em); }
rowGutter_col-11 { flex: 0 0 auto;
width: calc(91.6% - 1em); }
rowGutter_col-12 { flex: 0 0 auto;
width: calc(100% - 1em); }

.sm-1, .sm-2, .sm-3, .sm-4, .sm-5, .sm-6, .sm-7, .sm-8, .sm-9, .sm-10, .sm-11, .sm-12,
.md-1, .md-2, .md-3, .md-4, .md-5, .md-6, .md-7, .md-8, .md-9, .md-10, .md-11, .md-12,
.lg-1, .lg-2, .lg-3, .lg-4, .lg-5, .lg-6, .lg-7, .lg-8, .lg-9, .lg-10, .lg-11, .lg-12 {
flex: 0 0 100%; }

rowGutter_sm-1, .sm-2, .sm-3, .sm-4, .sm-5, .sm-6, .sm-7, .sm-8, .sm-9, .sm-10, .sm-11, .sm-12,
.md-1, .md-2, .md-3, .md-4, .md-5, .md-6, .md-7, .md-8, .md-9, .md-10, .md-11, .md-12,
.lg-1, .lg-2, .lg-3, .lg-4, .lg-5, .lg-6, .lg-7, .lg-8, .lg-9, .lg-10, .lg-11, .lg-12 {
flex: 0 0 auto;
width: calc(100% - 1em); }

.col-1, .col-2, .col-3, .col-4, .col-5 {
width: calc(100% - 1em);
}
rowGutter_col-1, rowGutter_col-2, rowGutter_col-3, rowGutter_col-4,
rowGutter_col-5 {
width: calc(100% - 1em);
}
}

/*----- SMALL SCREEN DEVICES -----*/

@media only screen and (min-width: 26em) {
.sm-1 { flex: 0 0 8.3%; }
.sm-2 { flex: 0 0 16.6%; }
.sm-3 { flex: 0 0 25%; }
.sm-4 { flex: 0 0 33.3%; }
.sm-5 { flex: 0 0 41.6%; }

```

```

sm-6 { flex: 0 0 50%; }
sm-7 { flex: 0 0 58.3%; }
sm-8 { flex: 0 0 66.6%; }
sm-9 { flex: 0 0 75%; }
sm-10 { flex: 0 0 83.3%; }
sm-11 { flex: 0 0 91.6%; }
sm-12 { flex: 0 0 100%; }

rowGutter sm-1 { flex: 0 0 auto;
width: calc(8.3% - 1em); }
rowGutter sm-2 { flex: 0 0 auto;
width: calc(16.6% - 1em); }
rowGutter sm-3 { flex: 0 0 auto;
width: calc(25% - 1em); }
rowGutter sm-4 { flex: 0 0 auto;
width: calc(33.3% - 1em); }
rowGutter sm-5 { flex: 0 0 auto;
width: calc(41.6% - 1em); }
rowGutter sm-6 { flex: 0 0 auto;
width: calc(50% - 1em); }
rowGutter sm-7 { flex: 0 0 auto;
width: calc(58.3% - 1em); }
rowGutter sm-8 { flex: 0 0 auto;
width: calc(66.6% - 1em); }
rowGutter sm-9 { flex: 0 0 auto;
width: calc(75% - 1em); }
rowGutter sm-10 { flex: 0 0 auto;
width: calc(83.3% - 1em); }
rowGutter sm-11 { flex: 0 0 auto;
width: calc(91.6% - 1em); }
rowGutter sm-12 { flex: 0 0 auto;
width: calc(100% - 1em); }
}

/*-----MEDIUM SCREEN DEVICES-----*/

@media only screen and (min-width: 40em) {
md-1 { flex: 1 0 8.3%; }
md-2 { flex: 1 0 16.6%; }
md-3 { flex: 1 0 25%; }
md-4 { flex: 1 0 33.3%; }
md-5 { flex: 1 0 41.6%; }
md-6 { flex: 1 0 50%; }
md-7 { flex: 1 0 58.3%; }
md-8 { flex: 1 0 66.6%; }
md-9 { flex: 1 0 75%; }
md-10 { flex: 1 0 83.3%; }
md-11 { flex: 1 0 91.6%; }
md-12 { flex: 1 0 100%; }

rowGutter md-1 { flex: 0 0 auto;
width: calc(8.3% - 1em); }
rowGutter md-2 { flex: 0 0 auto;
width: calc(16.6% - 1em); }
rowGutter md-3 { flex: 0 0 auto;
width: calc(25% - 1em); }
rowGutter md-4 { flex: 0 0 auto;
width: calc(33.3% - 1em); }
rowGutter md-5 { flex: 0 0 auto;
width: calc(41.6% - 1em); }
rowGutter md-6 { flex: 0 0 auto;
width: calc(50% - 1em); }
rowGutter md-7 { flex: 0 0 auto;
width: calc(58.3% - 1em); }
rowGutter md-8 { flex: 0 0 auto;
width: calc(66.6% - 1em); }

```

```

rowGutter md-9 { flex: 0 0 auto;
width: calc(75% - 1em); }
rowGutter md-10 { flex: 0 0 auto;
width: calc(83.3% - 1em); }
rowGutter md-11 { flex: 0 0 auto;
width: calc(91.6% - 1em); }
rowGutter md-12 { flex: 0 0 auto;
width: calc(100% - 1em); }
}

/*-----LARGE SCREEN DEVICES-----*/

@media only screen and (min-width: 48.1em) {
lg-1 { flex: 0 0 8.3%; }
lg-2 { flex: 1 0 16.6%; }
lg-3 { flex: 1 0 25%; }
lg-4 { flex: 1 0 33.3%; }
lg-5 { flex: 1 0 41.6%; }
lg-6 { flex: 1 0 50%; }
lg-7 { flex: 1 0 58.3%; }
lg-8 { flex: 1 0 66.6%; }
lg-9 { flex: 1 0 75%; }
lg-10 { flex: 1 0 83.3%; }
lg-11 { flex: 1 0 91.6%; }
lg-12 { flex: 1 0 100%; }

rowGutter lg-1 { flex: 0 0 auto;
width: calc(8.3% - 1em); }
rowGutter lg-2 { flex: 0 0 auto;
width: calc(16.6% - 1em); }
rowGutter lg-3 { flex: 0 0 auto;
width: calc(25% - 1em); }
rowGutter lg-4 { flex: 0 0 auto;
width: calc(33.3% - 1em); }
rowGutter lg-5 { flex: 0 0 auto;
width: calc(41.6% - 1em); }
rowGutter lg-6 { flex: 0 0 auto;
width: calc(50% - 1em); }
rowGutter lg-7 { flex: 0 0 auto;
width: calc(58.3% - 1em); }
rowGutter lg-8 { flex: 0 0 auto;
width: calc(66.6% - 1em); }
rowGutter lg-9 { flex: 0 0 auto;
width: calc(75% - 1em); }
rowGutter lg-10 { flex: 0 0 auto;
width: calc(83.3% - 1em); }
rowGutter lg-11 { flex: 0 0 auto;
width: calc(91.6% - 1em); }
rowGutter lg-12 { flex: 0 0 auto;
width: calc(100% - 1em); }
}

```

The is the CSS for the navigation options

```

/*-----Basic Bottom Navigation-----*/

nav {
max-width: 960px;
margin: 0 auto;
}

#nav-basic ul {
display: flex;
flex-direction: column;
align-items: center;

```

```

    list-style: none;
    padding: 5px 0;
}

#nav-basic ul li {
    align-self: stretch;
    text-align: center;
    border-bottom: 3px solid #383838;
    transition: .5s all;
}

#nav-basic li a {
    padding: 0 20px;
    color: #fff;
    text-decoration: none;
    font-weight: bold;
    text-transform: uppercase;
    letter-spacing: 0.1em;
    line-height: 2em;
    height: 2em;
    width: 100%;
}

#nav-basic li last-child a {
    border-bottom: none;
}

#nav-basic li hover,
#nav-basic li focus {
    color: #1c1c1c;
    border-bottom: solid 3px #ccc;
}

navfooter {
    max-width: 100%;
    background: #383838;
    color: #ffff;
}

nav-basic-header {
    position: relative;
    display: flex;
    justify-content: space-between;
    line-height: 70px;
}

logo {
    padding: 0 20px;
}

h1 logo a {
    text-decoration: none;
    font-weight: bold;
    color: #ffff;
    text-transform: uppercase;
    font-size: 1em;
}

.to_nav {
    background: #383838;
    border-left: solid 1px white;
    position: absolute;
    padding: 0 2.5rem;
    right: 0;
    top: 0;
    color: #ffff;
    display: block;
    font-size: 0.85em;
}

```

```

font-weight: 600;
text-decoration: none;
transition: 0.5s all ease;
}

to_nav:hover,
to_nav:focus {
color: #1c1c1c;
background: #ccc;
}

@media only screen and (min-width: 48em){
a to_nav {
display: none;
}

#nav-basic ul {
flex-wrap: wrap;
flex-direction: row;
}

#nav-basic ul li {
border-bottom: solid 3px #2E343A;
border-right: solid 1px white;
}

#nav-basic ul li:nth-last-child(1){
border-right: none;
}

mainheader { order: 1;}
navfooter { order: 2;}
mainContentWrapper { order: 3;}
mainfooter { order: 4;}

b2top {
display: none;
}

nav-basic-header {
justify-content: center;
}
}

/*
-----DROPDOWN NAV MENU-----*/

#logo {
line-height: 70px;
position: relative;
}

menuDown {
box-shadow: 0 3px 5px rgba(0,0,0,.15);
}

menuUp {
box-shadow: none;
}

#logo > h1 {
padding: 0 10px;
text-decoration: none;
font-weight: bold;
color: #000000;
}

```

```

text-transform: uppercase;
font-size: 2em;
}

#logo > #navToggle {
background: #383838;
border-left: solid 1px white;
position: absolute;
right: 0;
top: 0;
transition: 300ms all ease;
}

#logo > #navToggle:hover {
background-color: rgba(0,0,0, 1);
}

#logo > #navToggle > a {
color: #ffff;
display: block;
font-size: 0.85em;
font-weight: 600;
padding: 0 2.5rem;
text-decoration: none;
transition: 0.5s all ease;
}

#logo > #navToggle:hover > a {
color: #ffff;
}

.dropdown {
background: white;
display: none;
flex: 1;
transition: 0.5s all ease;
}

.dropdown > ul {
list-style-type: none;
}

.dropdown > ul > li {
align-self: stretch;
text-align: center;
border-bottom: 3px solid #383838;
transition: .5s all ;
position: relative;
}

.dropdown > ul > li:last-of-type {
border-bottom: none;
}

.dropdown > ul > li > a {
display: block;
color: rgba(0,0,0, .65);
font-weight: 700;
padding: 1.5rem 0;
text-decoration: none;
transition: 250ms all ease;
}

.dropdown > ul > li > a span.toggle {
background: #d8d8d8;
border-radius: 5px;
margin-left: .5em;

```

```

    color: #000000;
    font-size: 0.75em;
    padding: 2px 8px;
    text-transform: uppercase;
}

dropdown > ul > li > a span.caret {
    display: none;
}

dropdown > ul > li > nav {
    background: #383838;
    padding: .5em;
    border-radius: .5em;
    box-shadow: 0 2px 8px rgba(0,0,0,.6);
    display: none;
    flex: 1;
    overflow: hidden;
    position: absolute;
    right: 5%;
    width: 90%;
    z-index: 100;
}

dropdown > ul > li > nav > ul > li {
    align-self: stretch;
    text-align: center;
    border-bottom: solid 1px #383838;
    transition: .5s all;
}

dropdown > ul > li > nav > ul > li > a {
    color: rgba(255,255,255,.85);
    padding: 0 20px;
    color: #fff;
    text-decoration: none;
    font-weight: bold;
    text-transform: uppercase;
    letter-spacing: 0.1em;
    line-height: 2em;
    height: 2em;
    width: 100%;
    transition: 300ms all ease;
}

dropdown > ul > li > nav > ul > li > a:hover > a {
    background-color: rgba(0,0,0,.6);
    color: rgba(255,255,255,1);
}

@media all and (min-width: 48em) {
    #logo > #navToggle {
        display: none;
    }

    #logo {
        background-color: transparent;
        line-height: 90px;
        text-align: center;
    }

    #logo > h1 {
        color: #ffffff;
        text-transform: uppercase;
        font-size: 2em;
    }

```

```

}

dropdown {
  background-color: transparent;
  display: block;
}

dropdown ul {
  display: flex;
  flex-wrap: wrap;
  flex-direction: row;
  padding: 5px 0;
}

dropdown ul li {
  align-self: stretch;
  text-align: center;
  border-bottom: none;
  border-right: solid 1px white;
}

dropdown ul li:nth-last-child(1){
  border-right: none;
}

dropdown ul li a {
  padding: 0 20px;
  color: #fff;
  text-decoration: none;
  font-weight: bold;
  text-transform: uppercase;
  letter-spacing: 0.1em;
  line-height: 2em;
  height: 2em;
  width: 100%;
}

dropdown > ul > li > a span.toggle {
  display: none;
}

dropdown > ul > li > a span.caret {
  border-bottom: 4px solid transparent;
  border-top: 4px solid white;
  border-right: 4px solid transparent;
  border-left: 4px solid transparent;
  border-radius: 1px;
  content: "";
  display: inline-block;
  height: 0;
  margin: 0 0 0 .25rem;
  transition: 250ms all ease;
  width: 0;
  vertical-align: middle;
}

dropdown > ul > li:hover > a span.caret, dropdown > ul > li:focus > a span.caret {
  border-top-color: white;
  transform: rotate(270deg);
}

dropdown > ul > li:hover > nav {
  background-color: #d8d8d8;
  border-radius: 5px;
  box-shadow: 0 2px 8px rgba(0,0,0,6);
  display: flex;
  flex-direction: column;

```

```
    line-height: 3em;  
    right: -50%;  
    width: 196px;  
}
```