Introducing Responsive Web Design in Web Development Instruction

Jose Ortiz
California State University, Los Angeles
jarriol6@calstatela.edu

Arun Aryal
California State University, Los Angeles
aaryal@calstatela.edu

Abstract

Access to websites through mobile devices has steadily increased in the last decade. Along with the growing use of mobile devices, the web community has advanced responsive web design as an alternative to creating mobile-friendly websites. Responsive web design has become common practice in web development, as businesses of all types often require their websites to be mobile-friendly. Nonetheless, web development courses outside computer science (CS) majors often do not include responsive web design in their programs. This paper investigates how responsive web design can be taught in an introductory web development course for students with no prior computer programming background. This paper's contribution is to provide concise, simple, and self-contained explanations and examples that instructors can incorporate into an introductory web development course in non-CS majors.

Introduction

Access to websites through mobile devices has steadily increased in the last decade. This growth hit a new milestone in December 2019 by surpassing desktop browser access for the first time in North America (Statcounter, 2022). Along with the increasing use of mobile devices, the web community has advanced responsive web design. Responsive web design is an approach to creating mobile-friendly websites that automatically adapt to their viewing environment (Marcotte, 2010), providing an optimal viewing experience consistent across different devices. Responsive web design has become prevalent in modern websites (Al-Qallaf & Ridha, 2019; Oppenlaender et al., 2020), as businesses of all types often require their websites to be mobile-friendly.

The growing demand for web developers and mobile-friendly websites turns the spotlight on responsive web design as an essential skill in web development instruction. This argument is supported by the U.S. Bureau of Labor Statistics, which estimated that the employment growth for web developers is the highest among all computing occupations projected for 2014-2024 (U.S. Bureau of Labor Statistics, 2016). Teaching responsive web design develops skills critically needed in web development and provides an opportunity to raise awareness of technology trends and the impact of rapidly evolving IT on business. Knowledge of responsive web design is thus relevant for students in information systems majors pursuing careers in software development and managing information technology projects.

Web development instruction outside computer science (CS) majors to students with no prior programming raises challenges. Among the challenges is the multiplicity of technologies involved in web development. Working with more than one technology can be overwhelming for students without prior exposure to computer programming. For this reason, introductory web development courses often do not include responsive web design in their programs (Jin, 2017). As a result, websites "created by students often look outdated and clunky" (p. 115) and very different from the websites they access on their mobile devices and desktops in their daily lives.

This paper investigates how responsive web design can be taught in an introductory web development course for students with no prior programming background. The paper discusses further the need for responsive web design, current paradigms for building mobile-friendly websites, and their benefits and limitations. The paper then explains two approaches to implementing responsive designs: web standards-based techniques and using a front-end framework. The first approach leverages HTML and modern CSS-based techniques for implementing responsive design. The second approach uses the Bootstrap framework to achieve the same results, significantly reducing the complexity and coding effort. The paper concludes by discussing the use of front-end development frameworks in web development instruction. This paper's contribution is to provide concise, simple, and self-contained explanations and examples that instructors can incorporate into an introductory web development course in non-CS majors.

Mobile Web Development

Mobile Website Traffic

Access to websites through mobile devices has steadily increased in the last decade. Figure 1 shows that mobile browser access surpassed desktop browser access in December 2019 for the first time in North America. In June 2022 alone, mobile devices generated 52.93% of website traffic. Figure 2 shows the most performed activities on mobile browsers, with shopping listed as one of the most prevalent. These trends pose compelling reasons for businesses to ensure their websites are mobile-friendly, especially small businesses that rely heavily on the Internet to promote and sell their products.

The growing use of mobile devices to access websites has led to the invention of new techniques and practices in web design. Web design is the arrangement of content into visual representations or models that web developers can use as "blueprints" to implement websites and web applications (Dennis et al., 2015). The challenge that mobile access raises in web design is ensuring that the content of web pages fits and shows correctly on the "itty-bitty living spaces" (Krug, 2018) of mobile devices' screens. Adding to this difficulty is the variety of devices people use to access websites, ranging from smartphones and tablets to laptops and desktop computers. Furthermore, compatibility with mobile devices has also become relevant to search engines like Google, which favors mobile-friendly websites for its mobile search results. Since 2015, searches done on mobile devices have prioritized mobile-friendly websites where "text is readable without zooming, tap targets are spaced appropriately, and the page avoids horizontal scrolling" (Google Search Central Blog, 2016). Practitioners and learners willing to check how easily a visitor can use their web page on a mobile device can use Google's mobile-friendly test at search.google.com/test/mobile-friendly.

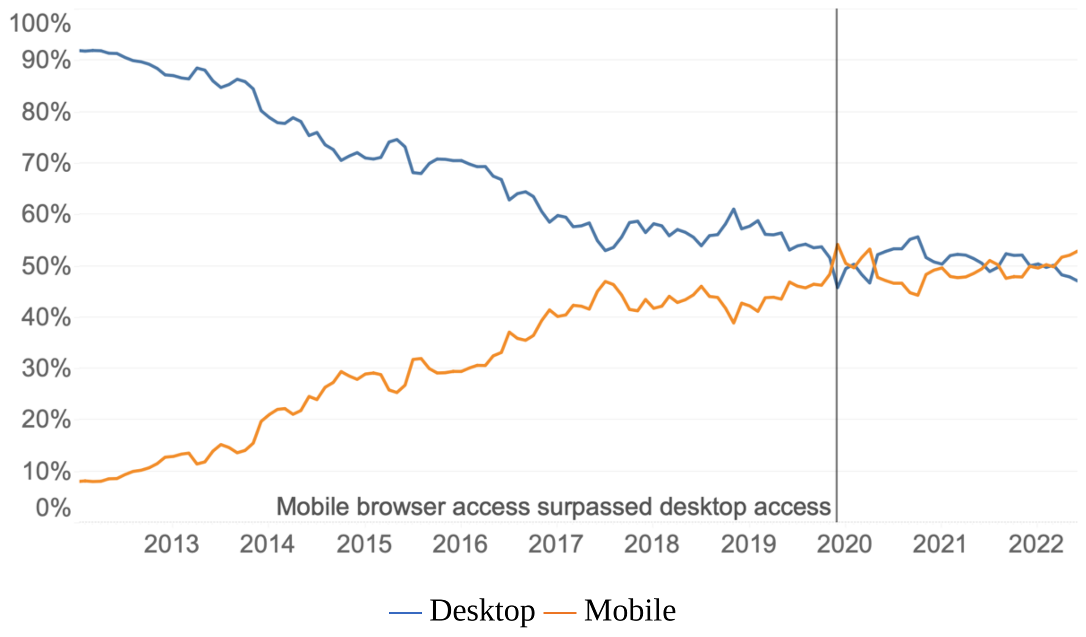


Figure 1 Desktop vs. Mobile Market Share in North America
January 2021 – June 2022

This chart uses data from Statcounter (June 2022). Desktop vs. Mobile Market Share North America as of June 2022. In Statcounter. Retrieved November 04, 2022, from https://gs.statcounter.com/platform-market-share/desktop-mobile/north-america

Approaches to Mobile Web Development

The web community has proposed several approaches and technologies to facilitate the design and development of mobile-friendly websites, i.e., websites designed to work on mobile devices with smaller screens. Approaches for building mobile-friendly websites fall into main two paradigms: separate websites and responsive web design (Cazañas & Parra, 2017).

Separate Websites

The separate websites approach consists of creating a different website for mobile access that is often accessed using a different URL prefixed with the letter "m" (for mobile), i.e., m.example.com. A typical business case for this approach involves websites where the user goals of the mobile site are more limited in scope than their desktop equivalent. Iglesias and Meesangnil (2011) provide an excellent case of this scenario in their inquiry into the design of a library website. m.youtube.com and m.facebook.com are well-known sites using the separate websites approach.

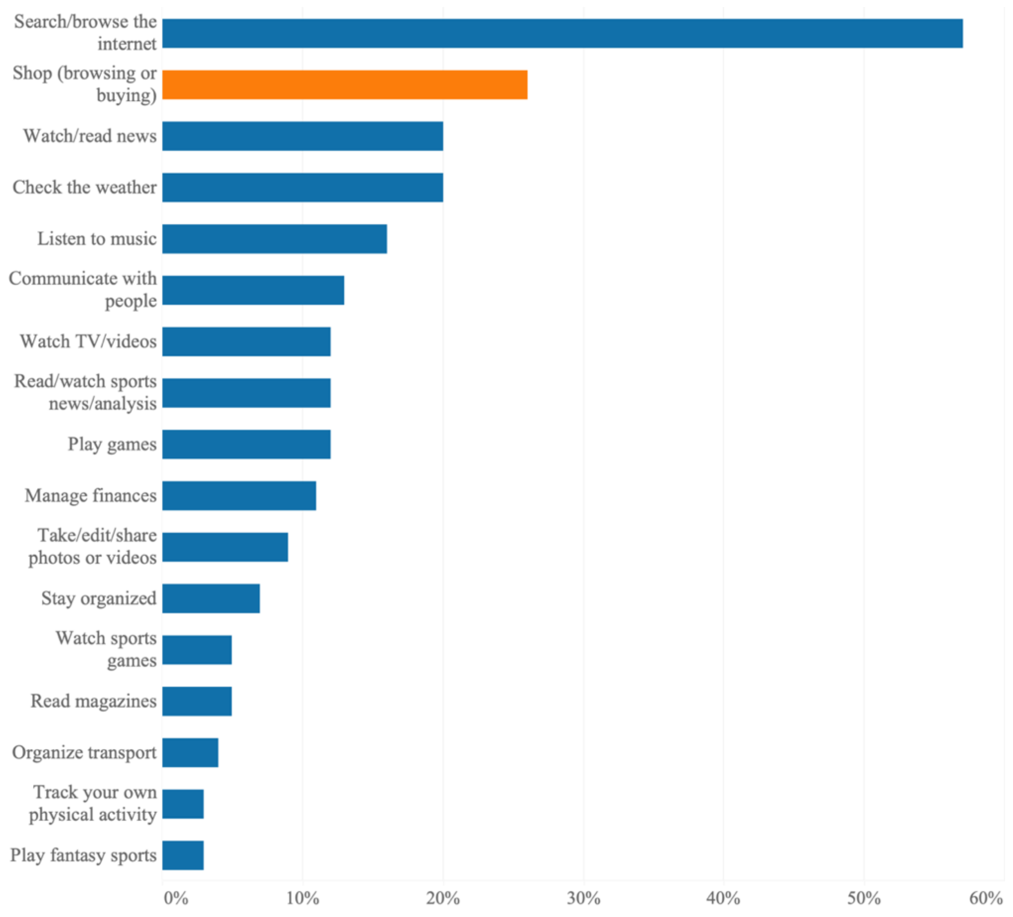


Figure 2 Popular Mobile Browser Activities

This chart uses data from Ipsos MORI (August 1, 2017). Most popular mobile browser activities according to smartphone users in the United States as of May 2017. In Statista. Retrieved November 04, 2022, from https://www.statista.com/statistics/267348/mobile-browser-activities-usa/

The benefits of the separate websites approach include tailoring the site to mobile-specific technical capabilities (Grlicky, 2011), such as location detection and device orientation commonly used by social media and user-generated content sites. Another advantage is improved performance since the browser does not need to perform adjustment operations to adapt the content to the mobile device's screen, as in responsive web design.

However, the separate websites approach has significant drawbacks. The main disadvantage is the cost of maintaining two separate sites. Web technologies and tools like content management systems facilitate the centralization of information through templates that can help to minimize this duplication (Grlicky, 2011). Maintaining mobile and desktop versions of the same site increases the effort needed to implement changes or add new functionality common to both versions since it requires writing two sets of front-end code.

Responsive Web Design

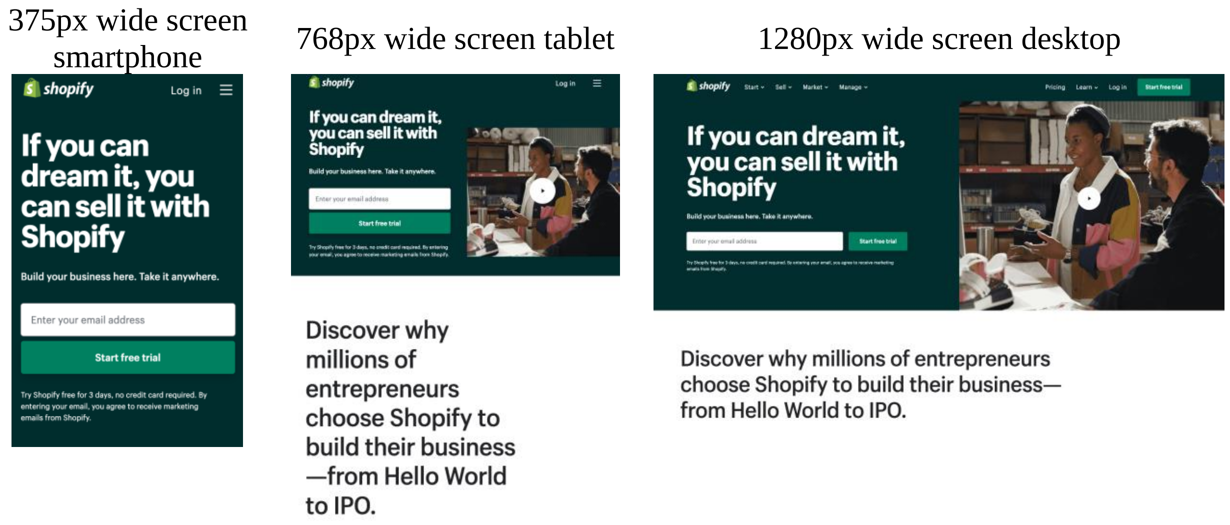
Responsive web design (responsive design for short) is an approach to creating web pages that automatically adapt to their viewing environment (Marcotte, 2010). A responsive website is, thus, a site where content displays correctly on any device regardless of screen size. In contrast to the separate websites approach, responsive design aims for an optimal viewing experience consistent across devices with different screen sizes, capabilities, and browser types.

Table 1 Standard screen sizes

Device	Size range
Smartphones	< 575px
iPads, tablets	576px to 767px
Laptops, small screens	768px to 991px
Desktops, large screens	⩾ 992px

Table 1 above presents size ranges commonly used in responsive design to describe the screens of different devices (Nimritee, 2021). It is worth noting that responsive design uses pixels as units to measure device screen sizes. These pixels are CSS "reference pixels" defined by the World Wide Web Consortium (W3C) specifically for web design and development and differ from those used to measure screen resolutions (Chien & Nyman, 2013). The World Wide Web Consortium is an international organization responsible for developing standards used in web design and development.

Error! Reference source not found. presents an example of a responsive website rendered on devices with different screen sizes. Note that on smaller screens, the page's content remains readable and is stacked along the screen's height, preventing the user from having to scroll to the sides of the screen (horizontal scrolling) to navigate the site.



If you can dream it, you can sell it with Shopify

Build your business here. Take it anywhere.

Try Shopify free for 3 days, no credit card required. By entering your email, you agree to receive marketing emails from Shopify.

Discover why millions of entrepreneurs choose Shopify to build their business—from Hello World to IPO.

Figure 3 Example of a responsive website

Marcotte (2010) introduced the term responsive design in 2010 as a solution to the limitations of approaches developed around that time to implement mobile-friendly websites. Responsive design brought about a new paradigm in web development that leverages standards-based web technologies to create web pages that are “not only more flexible, but more adaptive to the media that renders them” (Marcotte, 2010). Since its inception in 2010, responsive design has become a prevalent practice in web design with the continuous advancement of standards, techniques, and tools that facilitate the development of responsive websites (MDN Web Docs, 2022).

Since its conception, the main limitation of responsive design has been compatibility with older browsers (Marcotte, 2010). A survey by Almeida and Monteiro in 2017 reported that browser compatibility still being a major concern for responsive design among professional web developers. Responsive design’s compatibility issues stem from its reliance on *media queries* (Marcotte, 2010). *Media queries* is a technology that enables websites to inspect the physical characteristics of the device rendering the site, such as the size of the screen. *Media queries* are the basis of responsive design by facilitating the task of adapting web pages’ content to varying screen sizes. As popular browsers added support for media queries, this technology became a recommended standard by the W3C in 2012 (W3C, 2022b).

Implementing legacy browser support often requires JavaScript-based solutions (Marcotte, 2010), commonly known as polyfills. A polyfill is programming code that implements a feature in web browsers that do not support the feature natively. Polyfills add complexity that may affect a website’s performance and viewing experience. Thus, cross-browser testing is essential in the web development process to ensure the website performs as intended. The site “Can I use” (caniuse.com) is an excellent resource for checking browser support of media queries and other front-end web technologies used in responsive web design.

Responsive Web Design with CSS

Responsive design builds upon Cascading Style Sheets (CSS), one of the core web technologies alongside HTML and JavaScript. Using CSS, web designers can control any aspect of the website’s appearance, such as the size of images and text, color, spaces, and layout (W3C, 2022a). Responsive design combines three CSS-based techniques (Marcotte, 2010): fluid images, fluid grids, and media queries. The following section introduces methods for implementing responsive design techniques suitable for students with a basic understanding of HTML and CSS.

Fluid images

Responsive design uses the *max-width* CSS property set to “100%” on image elements (Marcotte, 2009b). This setting instructs the browser to adjust the image’s width so that it occupies the full width (100%) of the image container as long as the adjusted width does not exceed the image’s intrinsic size. Images with this property scale down if their containing element becomes narrower than the image’s intrinsic size but not larger, which may affect the image’s quality. The max-width property thus enables images on web pages to become smaller and fit on a smaller screen rather than overflowing it.

Figure 4 shows an example of the CSS max-width property to make images fluid in responsive design. To the bottom left, Figure 4 shows the result of the max-width property set to 100% on a 2000px image used as the web page's header displayed on an iPhone SE screen (375px wide). The result to the right shows the image in its original size (without the *max-width* property) on the same screen. The image overflows the browser's screen as its original size is larger than its container size (the *header* element, which has the same width as the browser's window in this case).

The max-width property provides a simple mechanism for enabling responsive behavior in web images. An important limitation of this technique is that while it enables large images to display correctly on small screens, the adjustment in the image's size occurs after the browser downloads the image. This may represent a problem for users of mobile devices with limited bandwidth who may be downloading images bigger than needed (Grigsby, 2015). The solution to this issue is to use media queries. Media queries can be written to dictate the browser to download images of a size appropriate to the browser's screen where it will be displayed and let the max-width property perform minor adjustments on the image's size if needed.

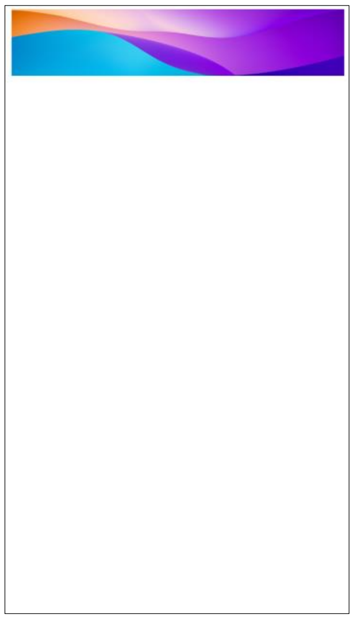
CSS

```
img {
   max-width: 100%;
}
```

HTML

```
<header>
   <img
      src="header-large.jpeg"
      alt="Header image with waves of colors">
</header>
```

Fluid image on an iPhone SE screen (375px)



Non-fluid image on an iPhone SE screen (375px)

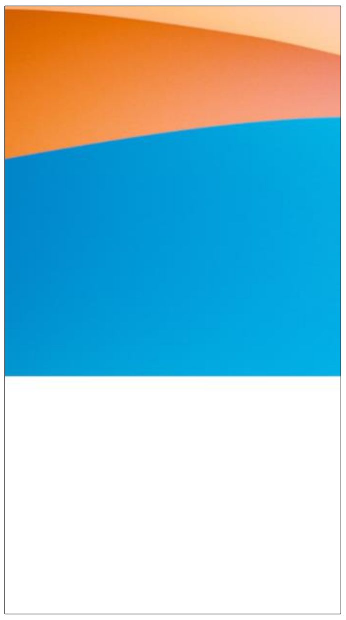
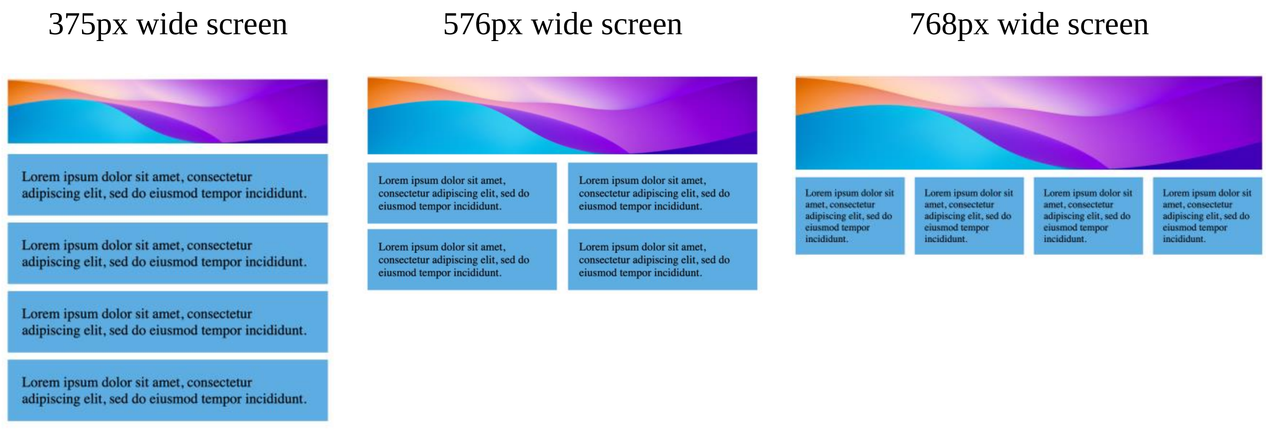


Figure 4 Example of fluid images

Fluid Grids

Like fluid images, a fluid grid provides various layouts enabling the website's content to adapt to the screen size of the device on which the site is displayed (Marcotte, 2009a). The web page's layout refers to how the page's content is visually arranged, commonly modeled as a structure of intersecting rows and columns, i.e., a grid.

The idea behind fluid grids is that devices with wide screens have more space to display their content across the screen's width (e.g., the content can take more columns on the grid). When displaying the same content on devices with smaller screens, the content is displayed along the screen's height to compensate for the lack of space (e.g., the content takes fewer columns on the grid). Figure 5 presents an example of a fluid grid rendered on screens with different sizes. The web page's layout changes from four columns and one row on a wide (768px) screen to two columns and two rows on a narrower (576px) screen, and on a 375px screen, the layout breaks again, stacking the content into a single column and four rows.



Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

Figure 5 Example of fluid grids

There are different CSS techniques for implementing fluid grids (MDN Web Docs, 2022). The float property is the legacy technique with the broadest browser support. Modern techniques include the flexible box layout, commonly known as flexbox, the grid layout, and the multi-column layout. The CSS multi-column layout makes it much easier to control the layout of a web page than other modern techniques and legacy techniques.

Figure 6 displays two layouts using the CSS multi-column layout technique by setting the *column-count* property on a container element (a *div* element in this case) to a value representing the grid's number of columns. Implementing a fluid layout with the multi-column layout requires detecting the screen's size to determine a value for the *column-count* property that best suits the viewers' screen. This is possible with media queries, as explained in the next section.

Media Queries

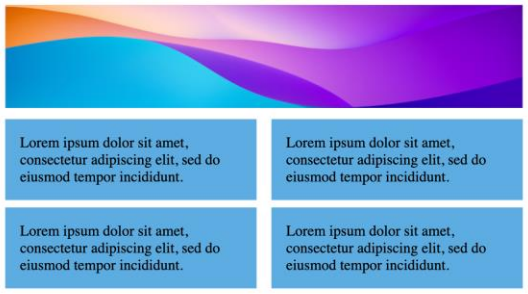
Media queries (W3C, 2021) are the cornerstone of responsive design (Marcotte, 2010). A media query provides a mechanism for testing specific device features where the web page is viewed, such as the screen's size.

Figure 7 below provides an example of a fluid grid using a media query that tests the width feature of the browser's screen. The CSS code asks the browser if the screen's horizontal width has a minimum width of 576px. This means the screen's width is equal to or greater than 576px. If the test passes—in other words, if we are viewing the web page on a device like a tablet—then the browser will display the content in two columns by applying the CSS rule setting the value of the *column-count* property to "2" of the container element. Otherwise, the browser ignores the CSS rule.

The values specified in each media query are known as breakpoints. A breakpoint, thus, defines a *point* on the screen's width at which the web page's design *breaks*, which means the layout changes to allow the page's content to fit the screen where it is viewed.

Layout with two columns

```
<div style="column-count: 2">
   <p> Lorem ipsum dolor ...</p>
   <p> Lorem ipsum dolor ...</p>
   <p> Lorem ipsum dolor ...</p>
   <p> Lorem ipsum dolor ...</p>
</div>
```



Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

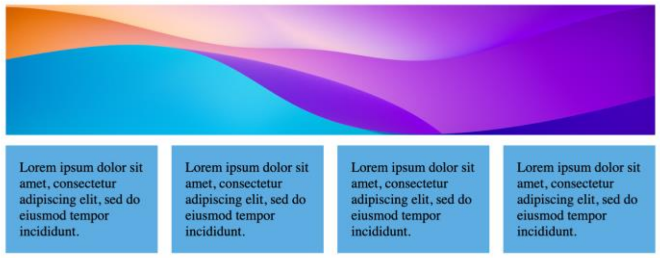
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

Layout with four columns

```
<div style="column-count: 4">
   <p> Lorem ipsum dolor ...</p>
   <p> Lorem ipsum dolor ...</p>
   <p> Lorem ipsum dolor ...</p>
   <p> Lorem ipsum dolor ...</p>
</div>
```



Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

Figure 6 Example of the CSS multi-column layout

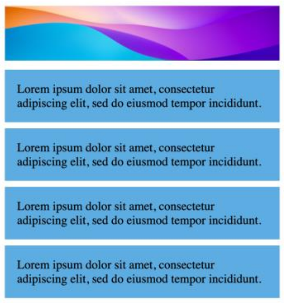
CSS

```
.container {
    /* Default layout:
    screen size < 576px */
    column-count: 1;
}
@media (min-width: 576px) {
    .container {
        column-count: 2;
    }
}
```

HTML

```
<div class="container">
   <p> Lorem ipsum dolor ...</p>
   <p> Lorem ipsum dolor ...</p>
   <p> Lorem ipsum dolor ...</p>
   <p> Lorem ipsum dolor ...</p>
</div>
```

Layout on screens < 576px wide


Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

Layout on screens ⩾ 576px wide

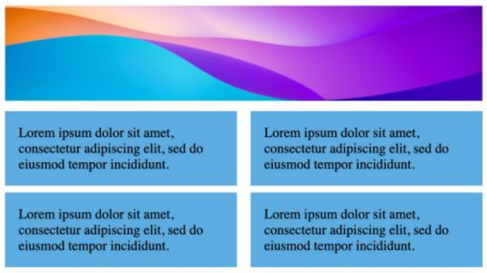

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

Figure 7 Example of fluid grids with CSS multi-column layout and one media query

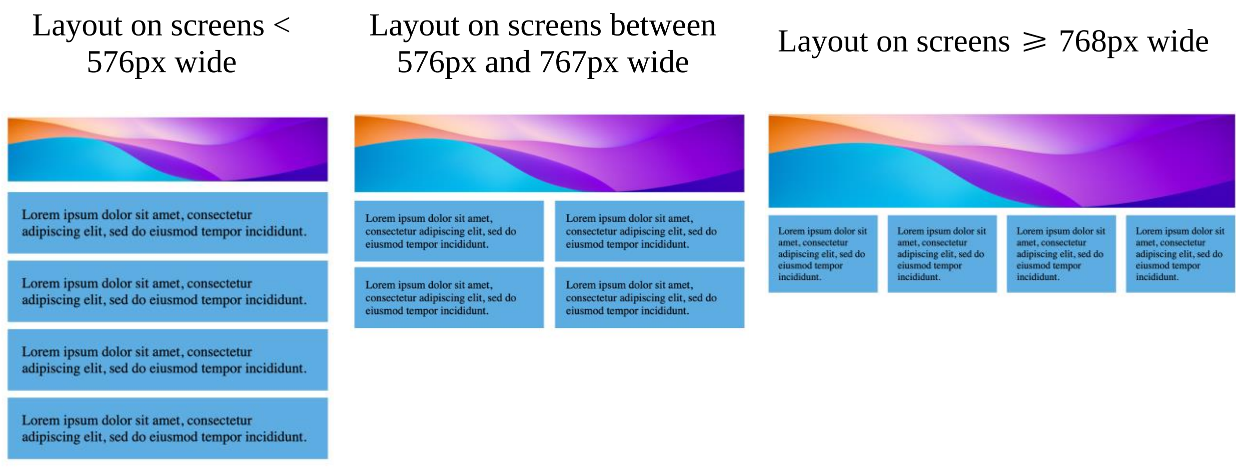
Adapting the web page’s content in Figure 7 to bigger devices, such as laptops and desktop computers, requires specifying an additional media query with a larger breakpoint, such as 768px (Figure 8), using Table 1 Standard screen sizes as a guide.

CSS

```
.container {
    /* Default layout for screen sizes < 576px */
    column-count: 1;
}
@media (min-width: 576px) {
    .container {
        column-count: 2;
    }
}
@media (min-width: 768px) {
    .container {
        column-count: 4;
    }
}
```

HTML

```
<div class="container">
   <p> Lorem ipsum
dolor ...</p>
   <p> Lorem ipsum
dolor ...</p>
   <p> Lorem ipsum
dolor ...</p>
   <p> Lorem ipsum
dolor ...</p>
</div>
```


Layout on screens < 576px wide
Layout on screens between 576px and 767px wide
Layout on screens ⩾ 768px wide

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

Figure 8 Example of fluid grids with CSS multi-column layout and two media queries

A final step in the implementation of responsive websites is the setup of the viewport metatag, as shown in **Error! Reference source not found.** (MDN Web Docs, 2022). The viewport corresponds to the web page's area that is visible to the user. By default, some browsers would shrink the web page's content to fit the current screen size. Setting the viewport's *initial-scale* property to "1" prevents such default behavior by instructing the browser to set the viewport's width to the device's screen width and scale the web page to 100% of its intended size. In this way, the browser displays the page at the mobile-optimized size intended by the page's responsive design.

HTML

```
<head>
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
```

Figure 9 Setup of the viewport metatag for enabling responsive design

Modern CSS technologies, like the multi-column layout, facilitate the implementation of responsive web design. However, responsive design requires novice learners to have not only knowledge of HTML but a thorough understanding of CSS and how to set up media queries with the appropriate breakpoint values. The web community has developed several frameworks that implement the CSS-based techniques required by responsive design into packages that can be easily integrated and customized into the construction of websites. The packages and reusable functionality provided by front-end frameworks can thus make web development easier for novice learners by reducing the complexity required to construct a responsive website.

Responsive Web Design with Bootstrap

Bootstrap is arguably the most popular front-development framework for developing responsive websites (W3Techs, 2022). Bootstrap simplifies the process of building responsive websites by providing reusable styles that can be easily incorporated into web pages. Bootstrap's styles consist of pre-defined CSS rules that can be directly applied to HTML elements using class selectors to implement responsive behavior. Figure 10 presents an example of the fluid images and fluid grids techniques implemented with Bootstrap.

Bootstrap provides the class *img-fluid* to make images fluid that applies the “max-width: 100%” property. To implement fluid grids, Bootstrap provides a grid system based on containers, rows, and columns. As shown in Figure 10, using Bootstrap’s grid system requires the following structure:

- All content in the grid is enclosed or contained within an element with the class *container*. The *container* class centers the grid on the web page.
- The columns in the grid are wrapped in an element with the class *row*. The *row* class adds a horizontal padding or gutter that controls the space between the columns.
- Each column in the grid is contained within an element with the class *col*.

The *container*, *row*, and *col* classes can be applied to any HTML element, such as division elements (<div>). Bootstrap’s grid system allows configuring the grid in several ways by refining the arrangement of the columns through suffixes added to the *col* class specifying breakpoints and column size based on a twelve-column template.

Bootstrap provides default breakpoints listed in Table 2 that can be used to adapt the number of columns in the grid to various screen sizes. As shown in Figure 10, the web page implements a fluid grid that behaves as follows:

- For extra small devices represented by the *xs* breakpoint (screen width less than 576px), the grid has only one column, which takes all twelve template columns.
- For medium devices represented by the *sm* breakpoint (screen width equal to or greater than 576px), the grid has two columns, each of which takes six template columns.
- For large devices represented by the *md* breakpoint (screen width equal to or greater than 768px), the grid has four columns, each of which takes three template columns.

Table 2 Default breakpoints in Bootstrap’s grid system

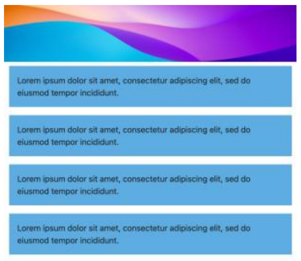
Breakpoint	**Range**
xs	< 576px
sm	⩾ 576px
md	⩾ 768px
lg	⩾ 992px

Bootstrap’s class suffixes for specifying breakpoints reduce the need to write media queries. Also, as shown in Figure 10, Bootstrap reduces the CSS coding effort, simplifying the process of implementing a responsive design. However, it requires a good understanding of the framework design and conventions, such as the grid system and the twelve-column template. Figure 11 illustrates Bootstrap’s twelve-column template used to adjust column width in the grid. The template shrinks and expands depending on the screen’s size to accommodate a fixed number of twelve columns. The twelve-column template allows adjusting column width at any given breakpoint.

HTML

```
<header>
        <img src="images/header-large.jpeg"
           alt=" alt="Header image with waves of colors"
           class="img-fluid">
</header>
<div class="container">
    <div class="row">
        <div class="col-xs-12 col-sm-6 col-md-3">
            <p class="rectangle">Lorem ipsum dolor sit... </p>
        </div>
        <div class="col-xs-12 col-sm-6 col-md-3">
            <p class="rectangle">Lorem ipsum dolor sit... </p>
        </div>
        <div class="col-xs-12 col-sm-6 col-md-3">
            <p class="rectangle">Lorem ipsum dolor sit... </p>
        </div>
        <div class="col-xs-12 col-sm-6 col-md-3">
            <p class="rectangle">Lorem ipsum dolor sit... </p>
        </div>
    </div>
</div>
```

Layout on screens < 576px wide



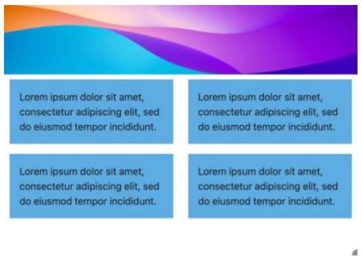
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

Layout on screens between 576px and 767px wide



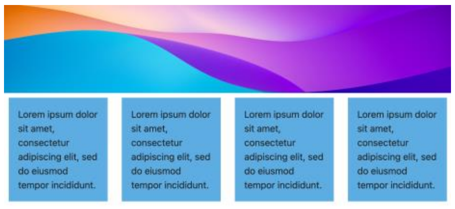
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

Layout on screens ⩾ 768px wide



Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

Figure 10 Example of fluid grids and fluid images with Bootstrap

The code shown in this example is available on GitHub at github.com/jarriol6/responsive-design.

Furthermore, Bootstrap follows a “mobile-first strategy” (Coyle, 2014) that emphasizes website access through mobile devices (Bootstrap, 2022). This design strategy is evident in the minimum thresholds used by most of Bootstrap’s default breakpoints. A breakpoint using a minimum width (e.g., *sm*) begins to apply at a specific breakpoint (e.g., 576px) and carries up through higher breakpoints (e.g., *md*, *lg*). This configuration allows for designing layouts that respond to mobile devices first and then adding additional styles for greater breakpoints that adapt the layout to larger devices like tablets and desktop/laptop computers (Coyle, 2014). Since mobile screens are small, a mobile-first strategy motivates designers to be more conscious about the amount of content to include in their web pages and work hard to determine what is essential; “what people need the most” (Krug, 2018, p. 143).

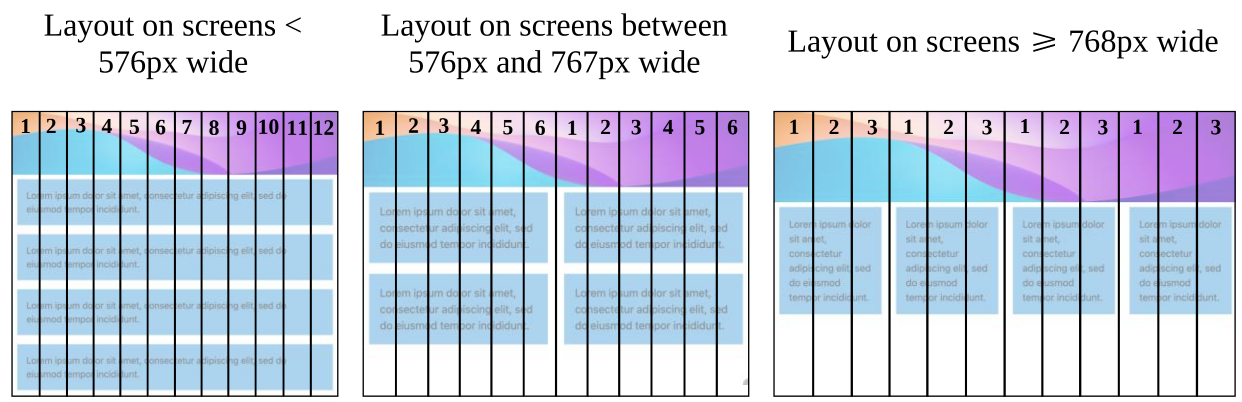

Layout on screens < 576px wide
Layout on screens between 576px and 767px wide
Layout on screens ⩾ 768px wide
1 2 3 4 5 6 7 8 9 10 11 12
1 2 3 4 5 6 1 2 3 4 5 6
1 2 3 1 2 3 1 2 3 1 2 3
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

Figure 11 The twelve-column template in Bootstrap’s grid system

Conclusions

This paper investigates how responsive web design can be taught in an introductory web development course for students with no prior computer programming background. It explains two approaches to implementing responsive designs: (1) using web standards-based techniques based on the CSS multi-column layout and media queries and (2) using the front-end development framework Bootstrap. These approaches can be introduced in the classroom after students have developed a solid understanding of HTML and CSS selectors. Implementing responsive designs using web standards-based techniques and Bootstrap allows students to compare two responsive implementation strategies and evaluate the benefits of using a front-end development framework.

Furthermore, working with a front-end development framework like Bootstrap also helps students to learn practices commonly used in software development, such as code reusability and self-learning (Jin, 2017). Using the same Bootstrap classes to implement different layouts and web pages gives students hands-on practice using existing code to create new software. Another essential practice that students can exercise by using Bootstrap is self-learning. Bootstrap maintains up-to-date online documentation with explanations and examples that students can refer to when working on their web development projects.

Besides its responsive classes, Bootstrap offers a wide range of classes for controlling other aspects of web design, such as typography and color, alongside reusable components, such as slideshows and dropdown menus, that incorporate responsive design by default. Leveraging existing code like Bootstrap’s pre-defined CSS classes and components also reduces the effort required to develop responsive websites, leaving room to focus on websites’ applicability and value to organizations.

References

Al-Qallaf, C. L., & Ridha, A. (2019). A Comprehensive Analysis of Academic Library Websites: Design, Navigation, Content, Services, and Web 2.0 Tools. *International Information & Library Review, 51*(2), 93-106.

Almeida, F., & Monteiro, J. (2017). The Role of Responsive Design in Web Development. *Webology, 14*(2).

Bootstrap. (2022). *Approach*. Bootstrap. Retrieved November 2022 from https://getbootstrap.com/docs/5.0/extend/approach/

Cazañas, A., & Parra, E. (2017). Strategies for Mobile Web Design. *Enfoque UTE, 7*(1), 344-357.

Chien, T., & Nyman, R. (2013). CSS Length Explained. *Mozilla Hacks*. Retrieved November 2022, from https://hacks.mozilla.org/2013/09/css-length-explained/

Coyle, J. (2014). How Does Progressive Enhancement Relate to Mobile-First Strategy? *Medialot*. Retrieved November 2022, from https://medialoot.com/blog/progressive-enhancement-web-design/

Dennis, A., Wixom, B., & Tegarden, D. (2015). *Systems Analysis and Design: An Object-oriented Approach with UML*. John Wiley & Sons.

Google Search Central Blog. (2016). *Continuing to make the web more mobile friendly*. Retrieved November 2022 from https://developers.google.com/search/blog/2016/03/continuing-to-make-web-more-mobile

Grigsby, J. (2015). Responsive Images 101. Retrieved November 2022, from https://cloudfour.com/thinks/responsive-images-101-definitions/

Grlicky, J. (2011). Approaches to Mobile Web Development Part 2 – Separate Sites. *Mozilla Web Development*. Retrieved November 2022, from https://blog.mozilla.org/webdev/2011/05/13/approaches-to-mobile-web-development-part-2-separate-sites/

Iglesias, E., & Meesangnil, W. (2011). Mobile Website Development: From Site to App. *Bulletin of the American Society for Information Science and Technology, 38*(1), 18-23.

Jin, K. H. (2017). Teaching Responsive Web Design to Novice Learners. Proceedings of the 18th Annual Conference on Information Technology Education, NY, USA.

Krug, S. (2018). *Don't Make Me Think!: Web & Mobile Usability: A Common Sense Approach to Web and Mobile Usability*. New Riders.

Marcotte, E. (2009a). Fluid Grids. *A List Apart*. Retrieved November 2022, from https://alistapart.com/article/fluidgrids/

Marcotte, E. (2009b). Fluid Images. *Unstoppable Robot Ninja*. Retrieved November 2022, from https://unstoppablerobotninja.com/entry/fluid-images/

Marcotte, E. (2010). Responsive Web Design. *A List Apart*. Retrieved November 2022, from https://alistapart.com/article/responsive-web-design/

MDN Web Docs. (2022). *Responsive design*. Mozilla. Retrieved November 2022 from https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Responsive_Design

Nimritee. (2021). How to Use CSS Breakpoints for Responsive Design. *Web Development*. Retrieved November 2022, from https://www.lambdatest.com/blog/how-to-use-css-breakpoints-for-responsive-design/

Oppenlaender, J., Tiropanis, T., & Hosio, S. (2020). CrowdUI: Supporting Web Design with the Crowd. *Proceedings of the ACM on Human-Computer Interaction*, *4*(EICS), 1-28.

Statcounter. (2022). *Desktop vs. Mobile Market Share North America*. https://gs.statcounter.com/platform-market-share/desktop-mobile/north-america

U.S. Bureau of Labor Statistics. (2016). *Computer and Information Technology Occupations*. https://www.bls.gov/ooh/computer-and-information-technology/home.htm

W3C. (2021). *Media Queries Level 4*. World Wide Web Consortium. Retrieved November 2022 from https://www.w3.org/TR/2021/CRD-mediaqueries-4-20211225/

W3C. (2022a). *Cascading Style Sheets*. World Wide Web Consortium. Retrieved November 2022 from https://www.w3.org/Style/CSS/Overview.en.html

W3C. (2022b). *Media Queries Level 3 Publication History*. World Wide Web Consortium. Retrieved November 2022 from https://www.w3.org/standards/history/mediaqueries-3

W3Techs. (2022). *Usage statistics of CSS frameworks for websites*. Web Technology Surveys. https://w3techs.com/technologies/overview/css_framework