Assignment 2

Contents

Directly writing raw, static HTML and CSS is feasible for smaller sites like the mythical one you designed for a01. But it's easy to see how designing a website from scratch can be difficult---especially for large, complex, dynamic sites containing many pages. And although every website you have ever used is ultimately expressed in terms of CSS and HTML, modern web developers typically have many tools at their disposal to help organize the code, reduce unnecessary duplication, and promote best practices. In this course, we're going to become familiar with a few of these tools.

In particular, frameworks are widely employed for assisting with web development. A framework is a structural code library that enforces an organizational scheme on the code that you write, with the goal of helping you to produce better software. Here are a few of the benefits of using a framework to organize your application:

- It speeds up the development process
- It makes your code easier to read
- It reduces the potential for bugs
- It prevents code replication
- It encourages you to conform to best practices
- It provides pre-written solutions to common tasks that you are likely to encounter

A framework is typically written for a particular language and is designed to help you write a particular type of software. For example, Angular, React, and Vue are examples of JavaScript frameworks designed to help you write organized JavaScript code for the browser. We'll learn more about these frameworks towards the end of the semester.

In this assignment we will be exploring Bulma, a CSS framework designed to help you organize the way you add style to HTML pages. By linking Bulma to your HTML pages and adding a few special classes to your HTML elements, you can quickly make a webpage that looks good and follows best design practices, all without having to write any custom CSS.

> While Bulma is a great CSS framework, it's not the only one out there. This blog post lists five other CSS frameworks that are also popular right now. When planning a new web app, picking a CSS framework can be difficult, and there are a lot of factors to consider. Ultimately, try to pick one that will allow you to rapidly produce well designed pages.

CSS frameworks are a quick and easy way to make pretty websites, but they're no substitute for understanding the underlying CSS rules that are being applied. For this course, it's important to have a working understanding of the fundamentals before turning to tools like Bulma.

1. Getting started

1. To get started, open Visual Studio Code and take a look at the a02 assignment directory. **Please pull the latest changes from github before starting (i.e. run `git pull`), tweeks to the assignment starter code may have been made after the start of the semester**.
2. A few HTML files have already been provided for you. Open these files and get a feel for the contents. You'll also notice that there are no CSS files included.
3. Next, open a terminal **in the a02 assignment directory** and initialize a new npm project for a02 using the `npm init` command (see the a00 instructions if you need help with this).
4. While you have the terminal open, also add Browsersync as a dependency to your new project (using the command `npm install browser-sync`).
5. Start Browsersync from your terminal (`browser-sync start -sw`) and see how the HTML files look when rendered in the browser. Since there is no CSS, they should look pretty plain.

2. Installing Bulma

In this assignment, we are going to add Bulma styles to the raw HTML provided in the assignment directory. The goal is to use Bulma to improve the way these pages look, all without writing a single line of CSS.

2.1. Downloading Bulma

Bulma is a CSS framework, which means it is essentially one giant pre-written CSS stylesheet. In order to start styling our pages with Bulma, we need to first link the HTML to the Bulma CSS. Recall from a01 that CSS stylesheets can be linked to HTML documents by adding a special `<link>` tag inside the `<head>` of the document.

However, this raises an important question: where should we store the giant Bulma CSS stylesheet, and what path do we put in the `<link href="" />` attribute in order to link to it? Generally speaking, there are three feasible possibilities:

1. Download the Bulma CSS file manually, save it in the a02 folder, and `<link>` to it with a relative path.

2. In the HTML file, `<link>` directly to an online version of the Bulma CSS file hosted by a Content Delivery Network (CDN).

3. Download the Bulma CSS file as a *dependency* using npm, and `<link>` to the downloaded file relatively from `node_modules`.

   A Content Delivery Network, or CDN, is a giant network of servers geographically spread across a large region---like the country, the continent, or the world. The servers in a CDN work together to deliver website data such as video, images, audio, html, etc. to users as fast and as reliably as possible. By spreading out the servers across the world, there is a greater chance that any given user will be geographically close to a CDN server. If the closest server in the network responds to a user's request, it's likely that the response will arrive even faster.

These three options are discussed in more detail on the official getting started with Bulma page. Using npm to install the package as a dependency is the recommended option, so that's what we will do for this assignment. In general, installing packages through npm is the best route when you need to add third party code to your web apps.

Therefore, to download Bulma as a dependency, run the following command from the terminal in your a02 folder:

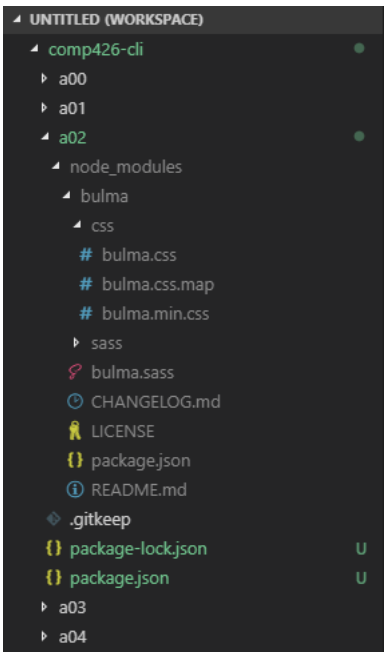
```
$ npm install bulma
```

Note: You may still have Browsersync running in your terminal from Section 1. Thus, to run `npm install bulma`, you either need to open a *new* terminal from the a02 folder (right click the a02 folder and select "Open in Terminal"), or you need to kill the Browsersync process with `Ctrl + C` to free up the existing terminal instance.

2.2. Finding the Bulma stylesheet

Now that Bulma has been downloaded, the next step is to connect the CSS to the HTML by adding `<link>` tags in the `<head>` of the HTML documents. But what URL should we use as the `href` attribute of the `<link>` tag? We need to look at the files that were downloaded by npm and find the correct CSS file.

Every time you `npm install` a new dependency, npm automatically downloads the requested files from the internet and puts them in a special folder named `node_modules`. This process occurred when you ran `npm install bulma`, meaning the Bulma files are now ready and waiting in the `node_modules` folder in your a02 folder. Open `node_modules` now and confirm that `bulma` has been downloaded. When you expand the folders, it should look something like this:



Inside of the new `a02/node_modules/bulma` folder is a list of files and folders that collectively make up Bulma. In particular, you'll notice a sub-folder named `css`, which contains three files:

- `node_modules/css/bulma.css`
- `node_modules/css/bulma.css.map`
- `node_modules/css/bulma.min.css`

The two files that end in `.css` are exactly what we're looking for! In fact, both `bulma.css` and `bulma.min.css` represent complete copies of the Bulma CSS source code, and either one would work for this assignment.

What is the difference between the two files? `bulma.min.css` is a *minified* version of Bulma, which means the CSS code was purposefully compressed to take up as few characters as possible. All newlines and whitespace are gone from the code, meaning it has a smaller file size and is therefore faster to transfer across the internet. Your browser doesn't need the extra whitespace, and is perfectly happy reading and understanding the minified version. However, minified code is generally much harder for humans to read!

2.3. Linking to Bulma in HTML

Now that we found the location of the Bulma CSS stylesheet, we can link it to the HTML files. The file we want is located at `node_modules/bulma/css/bulma.css` relative to the a02 root folder.

Add the following line inside the `<head>` of each HTML document in a02:

```
<link rel="stylesheet" href="node_modules/bulma/css/bulma.css" />
```

Make sure to place it above `<link rel="stylesheet" href="custom_style.css"/>`

2.4. Revisit rendered pages

Just by linking to Bulma, the provided HTML pages already look much better! Go ahead and take a look. If Browsersync is not running, start it up again and head to http://localhost:3000 on Google Chrome to see the rendered version.

Bulma automatically adds default styles to your page that affect little things like the default font family, font colors, font sizes, heading sizes, paragraph spacing, etc. Though subtle, these changes are important! They were carefully chosen by the Bulma team to reflect recommended best practice for web design.

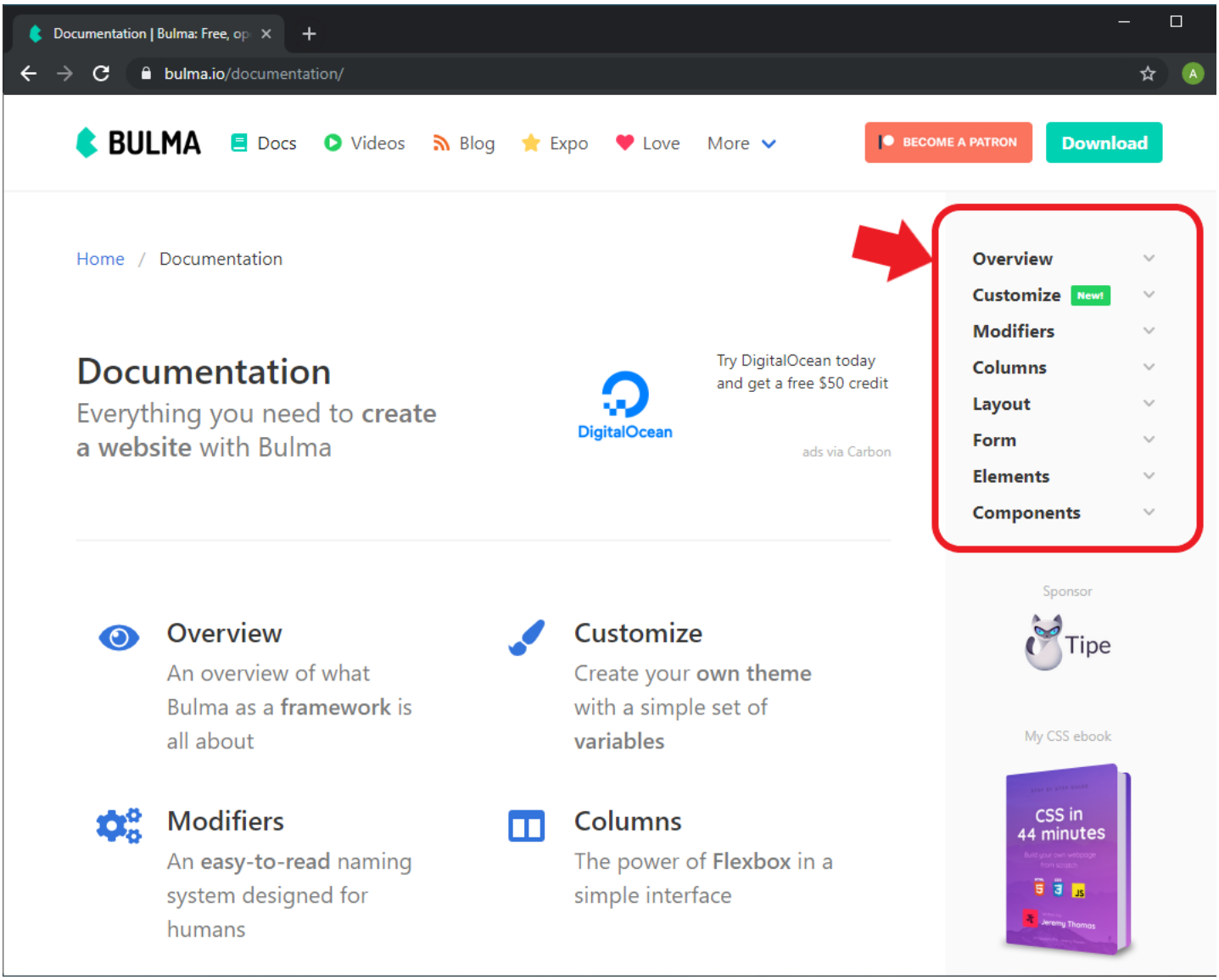
> If you're interested in learning more about best practices for good web style and design, there are lots of online resources available on the topics page. In particular, typography is the art of formatting text to be practical, effective, readable, and stylish.

3. Adding Bulma classes

The default styles added to your page by linking to Bulma are just the beginning. The real power of Bulma comes with its CSS classes. Bulma defines and styles hundreds of little utility CSS classes for use in your HTML. Need to change the background color of a `<div>` and make it yellow? Bulma pre-defines a CSS class for that. Need to add a red glow to a form input field when the user types an invalid value? There's a class for it. What about laying content side-by-side? Or making a table pretty? Or increasing the size of the text in a particular area? Or changing the font-family? Or giving buttons different colors? Bulma has classes for all of these too.

Bulma offers too many CSS classes to cover in this guide. However, you should become familiar with the official Bulma documentation, which is an excellent resource for getting to know your way around the framework and even gives code examples demonstrating what each CSS class does.

Go to the Bulma documentation and click through the menu on the right side of the page (see screenshot below). For the next part of the assignment, you will use Bulma classes to improve the design of the provided a02 HTML document.



4. Submission requirements

For this assignment, your task is to correctly add Bulma classes to the `a02/index.html` HTML document to make it look exactly like the screenshots in Section 4.1 below. You may not change the HTML elements on the page at all, except for adding Bulma classes. You also may not add any custom CSS styles.

You are required to reproduce the style from the screenshots exactly. To make this easier, there is a class "wordbank" in Section 4.2 containing all the classes you'll need to use. You'll also notice that as a hint, some of the elements in the HTML file already have their classes defined; there is no need to add or remove classes from these elements.

> Some IDEs provide automatic autocomplete when you start typing Bulma classes. This functionality is built in to Webstorm, but if you want Bulma class autocompletion on Visual Studio Code, you'll have to add it as an extension.

4.1 Website Screenshots

Here are screenshots of the website that you are trying to create. To get full credit on this assignment, your website must automatically match all these images by correctly using the Bulma classes listed in the next section alone.

- 4k
- 1080p
- narrow
- mobile

4.2 Classes to Use

Notice how this isn't just a list of classes used, but a list of the classes that were used to style an html element. So there is no mixing and matching necessary among the rows of this list. For example, when you see `subtitle has-text-grey has-text-weight-light` you can assume that this is styling a single html element. In this case it styles the subtitle on line 81. You won't see combinations other than the ones listed below.

Class List	Documentation Section
button is-dark	Buttons
checkbox	Forms
column	Columns/Layout
columns	
columns is-multiline justify-center	
container	Layout
content	Elements -> Content
content has-text-centered	
content has-text-right	
control	Form
field	Form
has-text-weight-bold	Modifiers -> Typography
hero is-dark is-fullheight	Layout -> Hero
hero-body	
input	Form
is-active	
is-capitalized	
label	
notification is-danger	Elements -> Notification
section	Layout -> Section
section has-background-white	
select	
subtitle	
subtitle has-text-grey	
subtitle has-text-grey has-text-weight-bold	
subtitle has-text-grey has-text-weight-light	
subtitle has-text-grey has-text-weight-normal	
subtitle has-text-grey is-italic	
tabs is-medium is-centered	
title	
title has-text-info is-family-secondary	
title has-text-weight-bold	
title has-text-weight-light	
title has-text-weight-normal	

5. More about npm *(Optional)*

This section introduces you to a few new npm commands. While this material isn't required for the assignment, it may come in handy in the future!

5.1. Updating dependencies

The dependencies that you add to your application may change over time. Each dependent package is owned and maintained by a different developer or team of developers, and many of these packages are actively being improved. This means that occasionally the maintainer of a package may release a new version of the code with bug fixes, new features, or code improvements. Every once and a while, you may want to make sure that your app is using the newest version of the dependencies. Luckily, npm makes this easy. Simply run the following command from a terminal pointed at your app's root folder.

```
$ npm update
```

The process may take a few seconds or minutes to complete, but when its done all of your dependencies will be up to date!

5.2. Removing a dependency

If you ever encounter a situation where you have installed a dependency that you later realize you do not need, you can use the `npm uninstall` command to remove the package both from the `node_modules` folder and from the `package.json` file.

```
$ npm uninstall package-name-here
```

5.3. About `package.json` and `package-lock.json`

We talked a little about `package.json` back in a00. It's a special configuration file created when you run `npm init`, and it contains various information about your project. Inside `package.json` is a list of all the packages that your app depends on. This list is managed by npm, and it gets updated whenever you run `npm install` or `npm uninstall`. You can open `package.json` and view the current dependency list at any time. In a02's `package.json`, you should see `bulma` and `browser-sync` listed as dependencies.

You may have noticed another file also appear in your project directory: `package-lock.json`. This is another special configuration file created by npm and used to manage your app's active dependencies. In particular, `package-lock.json` exhaustively lists the *version* of every package used by your app. This file comes in handy when you develop on multiple different computers, because it ensures that every system shares the exact same version of the exact same dependencies. Without `package-lock.json`, it would be possible for different machines to run different versions of the same dependencies, potentially causing inconsistencies at runtime.

> `package-lock.json` is updated when you run `npm update` to reflect the new versions of updated dependencies.

5.4. Local vs global npm packages

So far, we have used the `npm install package-name-here` to install new packages to our `package.json` file. *Packages installed in this way are installed locally to the app in the current directory*. This means that **the folder at which your terminal points matters a lot**, because it determines (1) where the dependency will be installed and (2) which `package.json` it will be attached to.

There is also a way to *globally* install a package:

```
$ npm install -g browser-sync
```

The `-g` in this command tells npm to install the `browser-sync` package to a special, hidden, system-wide `package.json`. Running the command like this will **not** install the package to the local `node_module`, or update the local `package.json`. The benefit of this approach for packages like Browsersync is that you can install them one time globally and then access them from anywhere.