Faculty of Computer Science, Dalhousie University *8-Feb-2017*

CSCI 2133 — Rapid Programming Techniques for Innovation

Lab 2: Python/Django 2 (We Built this City on CSS)

Lab Instructor: Colin Conrad
Location: Shiftkey Labs
Time: Wednesday, 17:30–19:00

Python/Django 2: We Built This City on CSS

Welcome to part 2 of the Shiftkey Labs Django Tutorial. Last week we learned the basics of Python and applied it to the Twitter API. This week we are going to focus on the other side of web applications ... the front end. This week's session is a big different from the other weeks in this series, as it is the only week that does not focus on the Python programming language. Rather, we will exclusively focus on building a mock of our application using HTML/CSS and JavaScript. By the end of the day today you should know just enough about HTML/CSS and JavaScript to build the application.

In the recent past, building beautiful and stable websites was a painstaking task. Today with the advent of HTML5 it has become easier to build robust and interactive pages. Most web developers use libraries that make difficult tasks easier and do a lot of heavy lifting. Some CSS and JavaScript libraries are built with a particular design philosophy in mind, and are optimized to deliver websites that incorporate these philosophies.

The design of our application will follow the Material design philosophy, and will use the Materialize CSS library to achieve this. You can read more about the Material design philosophy here `http://material.io` and about the Materialize CSS library here `http://materializecss.com`. Material design is a philosophy and design language created by Google, and is used in many of their mobile applications such as Google Fit or Google Keep. Much like Twitter's Bootstrap library, Material uses a grid system to organize objects, and visuals that are based on Google's "card" concept. Given that our Twitter application visualizes a number of distinct Twitter entities, it is well suited to Material's card design.

In order to function, the Materialize CSS library draws jQuery, one of JavaScript's most popular libraries. Though jQuery is now over 10 years old, but continues to be one of the most popular JavaScript tools. JQuery is optimized to manipulate the document object model (DOM) of a website, and allows a user to make changes to the site's structure in (much) fewer lines of code than raw JavaScript. We will be using jQuery to perform basic functions such as allowing Materialize's features to open, close or render. Refer back to the website sample and click through the cards to see jQuery in action `femto.cs.dal.ca/profileGrabber`

By the end of the day today you will have successfully created "home" and "discover" pages of the web application.

Step 1: Investigate the Folder

Let's start by opening the folder. You should see the following:

```
/assets
 -style.css
 -twitter.svg
discover.html
index.html
```

`week2-challenge.html`

For those who have experience with web development, you know that `index.html` is the file that is rendered when the url is resolved, while `discover.html` contains the html file for the discover page. I have renamed the collection.html file to `week2-challenge.html`. Yes, this week's challenge will be to create your own version of the collection page! The assets folder contains two files that support the html. The `style.css` file contains custom CSS features that modify our application, while the `twitter.svg` file is an image that is rendered by the `index.html` file. Open the index.html file in your web browser to see it in action.

While the picture of the twitter bird is contained in the assets folder, you will notice very quickly that the index.html file contains a large number of visual features that are not contained in the folder, including icons, interactive features and the like. These are actually rendered by the Materialize CSS library, rather than through the local files contained in the assets folder. We will explore how it does this later.

Now open the discover file in your web browser. The page largely renders empty html with a minimal number of css features, most notably a font type. We are going to clean up this file and make it look like `http://femto.cs.dal.ca/profileGrabber/discover.html`.

Step 2: Fix the Header

Open the `discover.html` file in your text editor. Since the Materialize CSS is not rendering, we are going to start by investigating the headers to make sure the files are included. Locate the ¡head¿ element and investigate.

```
<link href="http://fonts.googleapis.com/icon?family=Material+Icons"
rel="stylesheet">
<link rel="stylesheet" href="">
<link rel="stylesheet" type="text/css" href="assets/style.css"/>
```

These links make references to the CSS files that support the html. The first link brings the page to an external stylesheet designed for rendering Material icons. These are the styles that are used in the front page, and on the submit button! The third link directs us to a local stylesheet at assets/style.css. This was the stylesheet that we saw in the folder earlier.

The important thing is the second link, which is missing its href. Without a reference to the relevant CSS stylesheet, the html document is not able to proper render the materialize features that are written in the html body. Change this to the following:

```
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/
materialize/0.98.0/css/materialize.min.css">
```

Once you have changed this, open the page in your web browser. You will notice that the html populates with Materialize magic with just one line of code! Materialize, like many libraries, do not need to be accessed locally, and can be accessed by a web browser through a Content Delivery Network (CDN). CDN's help make the web page code maintainable, though it is always advisable to download local copies of these resources when you bring a page into production.

Step 3: Fix the Menu

We may have a nice-looking website, but there are still a few things wrong with the design. The first thing you may notice is that the menu colours are different from the rest of the application, and that the links are broken. Let's work on these. Take a look at the nav element:

```
<nav class="purple lighten-4">
      <div class="nav-wrapper container">
        <a href="/profileGrabber/" class="brand-logo">Twitter
                Profiler</a>
...
```

Clearly, the reason why our nav bar is purple (instead of blue) is because the nav has a `class=''purple lighten-4''`. Materialize makes it easy to define the colours of DOM objects using CSS classes. The class of purple naturally tells Materialize that you want this DOM object to be purple. The class of lighten-4 further tells Materialize to transform the purple colour into a lighter pattern. Using this method, it makes it very simple to define colour schemes that match the Material design philosophy. Let's change it to be consistent with the rest of the application, by making it blue.

```
<nav class="blue">
```

That's better. However, we are not finished yet. You will notice that the nav element is structured in a very specific way. Let's look at it in more detail.

```
      <div class="nav-wrapper container">
        <a href="/profileGrabber/" class="brand-logo">Twitter Profiler</a>
        <a href="#" data-activates="mobile-demo" class="button-collapse"><i class="materi
        <ul id="nav-mobile" class="right hide-on-med-and-down">
          <li><a href="/profileGrabber/index.html">Home</a></li>
          <li><a href="chinchillas.html">Discover</a></li>
          <li><a href="not-trolling.html">Collection</a></li>
        </ul>
        <ul class="side-nav" id="mobile-demo">
          <li><a href="/profileGrabber">Home</a></li>
          <li><a href="discover.html">Discover</a></li>
          <li><a href="collection.html">Collection</a></li>
        </ul>
      </div>
```

The menu items are clearly defined in the unordered list (`<ul>` and `<li>` tags) however they are directing to pages that do not exist. Let's try to fix this.

```
      <div class="nav-wrapper container">
        <a href="/profileGrabber/" class="brand-logo">Twitter Profiler</a>
        <a href="#" data-activates="mobile-demo" class="button-collapse"><i class="materi
        <ul id="nav-mobile" class="right hide-on-med-and-down">
          <li><a href="index.html">Home</a></li>
          <li><a href="discover.html">Discover</a></li>
          <li><a href="collection.html">Collection</a></li>
        </ul>
```

We changed the links in the `<li>` tags, so it should now link properly on desktop. There is a second set of links below these that should be changed as well. These links are define the side menu, which appears when a mobile device accesses the site. The mobile menu is rendered when the `materialize.js` file detects a mobile device, and follows the instructions defined by the `<a href=''#'' data-activates=''mobile-demo''` element. These element features basically tell the regular menu to disappear and the mobile side-menu to appear. Let's make these changes.

```
        <ul class="side-nav" id="mobile-demo">
          <li><a href="/profileGrabber">Home</a></li>
          <li><a href="discover.html">Discover</a></li>
          <li><a href="collection.html">Collection</a></li>
        </ul>
```

That should complete the menu. Save your changes and refresh the page in your browser. The color and links will have changed, which is good. However, if you try to investigate the page using a mobile device, the menu will not yet render. We will fix this later.

Step 4: Buttons and Grids

Though the page is starting to look nice, the button is messed up and the DOM features are out of order. See how `Fake Donald Trump` renders below the input element instead of beside it? Let's look at how Materialize renders the DOM and fix it accordingly, starting with the button.

```
<button class="" type="submit" name="action">Submit
  <i class="material-icons right">send</i>
</button>
```

Currently the button does not follow the Material design pattern and actually looks remarkably like the default HTML button. However, despite its flaws, the button *does* correctly render a materialize icon, contained in the `<i class='...'>` element. This is how we render material icons contained in the CSS library. Very handy for making things look nice. Let's try to make out button look nice as well by adding some effects, such as `waves-effect waves-light btn red` which will make it look nice.

```
<button class="waves-effect waves-light btn red" type="submit" name="action">Submit
  <i class="material-icons right">send</i>
</button>
```

Simple enough. What about Fake Donald Trump? Like Bootstrap CSS, Materialize uses a grid system to make it easier to organize content. You can think of the grid as containing 12 columns and an infinite number of rows, which can be defined by you. If you add elements to columns that add up to less than 12, then the elements in each column will appear side by side. Using an example, we can have contents of two `<div class='col m6></div>` elements, or four `<div class='col m3></div>` elements. However, we cannot have five size 3 elements side by side... Materialize would render the 5th element below the others.

Looking at our code, we can see the problem. Though the `<h4>Influencer:</h4>` is contained in a column, Fake Donald Trump is not. Let's fix this.

```
<div class="row">
      <!-- Here's a missing column. Try adding "col m3"-->
      <div class="col m3">
        <h4>Fake Donald Trump</h4>
        <span style="font-weight:bold;">@fakeDonaldTrump</span>
        <span>Murica</span>
      </div>
      <div class="col m3">
      <h4>Influencer: </h4>
```

Adding the `col m3` will make the page render properly. It is also worth noting that the `m` designates an instruction to render this way if being viewed on a medium sized screen. Mobile devices will render the columns differently.

Step 5: Modals and JavaScript

In the original version of this application, the user had the option of clicking on the green button and receiving information about Twitter influencers. Currently our web page does not do this. If we look at the code, we will can see that the green button contains features for a special DOM object called `modal` which is revealed when the button is pushed.

```
<div id="modal1" class="modal">
    <div class="">
      <h4>Influencers</h4>
      ...
```

The `modal` class is recognized by the Materialize JavaScript as a special class, while the `id=''modal1''` as a unique id for targeting. Currently the `<div class=''modal-content''` is missing. Let's adjust accordingly.

```
<div id="modal1" class="modal">
    <div class="modal-content">
      <h4>Influencers</h4>
      ...
```

If we click the button now, the modal will still not render. Let's try to find out why. Open the JavaScript console on your web browser (on Chrome right click, "inspect element" and "console". If you haven't seen this before, the browser console is the main tool for troubleshooting JavaScript. Though it is outside the scope of this tutorial series, becoming competent with the console is strongly recommended if you plan to develop web applications.

The console tells us that there are two JavaScript errors, one with `materialize.js` and another with our html file. We can first troubleshooting by making sure we have the correct JavaScript library configured. If we scroll to the bottom of the document we can find the script files.

```
  <script type="text/javascript" src=""></script>
    <script
  src="https://cdnjs.cloudflare.com/ajax/libs/materialize/0.97.8/js/materialize.min.js"><
```

These script files are used by Materialize to render the JavaScript correctly on the page. The second script element references the materialize.js file, which *must* complement the css library in order to function properly. However, the first script should reference jQuery, on which Materialize is dependent. Let's configure the script to reference a jQuery CDN.

```
<script type="text/javascript"
src="https://code.jquery.com/jquery-2.1.1.min.js"></script>
```

This will tell our file to properly configure jQuery. However, this is not yet enough. Below these script elements is another element which needs to be complete.

```
<script>
      $(document).ready(function(){
      });
</script>
```

This is a custom jQuery script which is needed to access the modal. Refer to the Materialize docs to find out what is missing. We can access these at `http://materializecss.com/modals.html`. If we scroll to the bottom of the page, there is some code to initialize the modal tool. Let's add it to the `<script>` element.

```
<script>
      $(document).ready(function(){
        $(' .modal').modal();
      });
</script>
```

This short code tells `materialize.js` to configure the instructions for modal and include them in this page. Save your work, refresh your browser and click the button. The modal will appear! There you have it! We have used JavaScript and Materialize CSS to make a fine looking web page. You can use Materialize in your other web development work. Though it is a very clean and powerful library, I strongly recommend that you check out other resources such as Bootstrap CSS, which is arguably the world's most popular CSS library `http://getbootstrap.com/css/`. See you next week!

WEEK 1 CHALLENGE

If you are ambitious, or are in CSCI 2133, I have another challenge for you. You will find the `week2-challenge.html` file in this folder. You will notice that the original web application also had the `collection.html` page, which is missing from this src folder. I wonder whether you can create a similar file, but with your own flare! The original file can be found at `http://femto.cs.dal.ca/profileGrabber/collection.html`. Can you create something similar, yet with an additional Materialize CSS feature? You will need to read the Materialize CSS docs to do this. Good luck!