WeatherScraper class written for DataScrape/DataApp/viewmodels.py

```
class WeatherScraper:

    def __init__ (self, zip):
        self.zipcode = zip

    #  Use Selenium to navigate to Weather.gov site.  The specific browser driver location
    #  required for instantiation is provided as a path in the call to Chrome().

        browser = webdriver.Chrome('DataApp/bin/chromedriver.exe')
        browser.get('http://weather.gov/')

        #  Upon landing at Weather.gov, the "Local forecast by" search box must be clicked to
        #  remove default "Enter location" text before sending the user's zipcode.
        browser.find_element_by_name('inputstring').click()
        browser.find_element_by_name('inputstring').send_keys(self.zipcode)

        #  Wait a while until the "Local forecast" search returns with alternate results for the
        #  user's zipcode before clicking the "Go" button to actually begin the search for local data.
        time.sleep(1)
        browser.find_element_by_name('btnSearch').click()

        #  Again wait a while until the page refreshes from the local data search before getting the
        #  URL info from the results page.
        time.sleep(1)
        url = browser.current_url

        #  Open the URL above for local forecast and get the HTML to parse with Beautiful Soup 4 (BS4)
        with urllib.request.urlopen(url) as response:
            page = response.read()
        soup = bs(page, 'html.parser')

        #  Close the browser opened for Weather.gov.
        #  ToDo:  Can the whole operation of opening the browser be made silent, so the user doesn't see it?
        browser.quit()

        # BS4 allows easy extraction of local temperature data by looking at a unique class identifier.
        self.temp = soup.find(class_='myforecast-current-lrg').get_text()

        #  For the humidity and last update info a slightly more sophisticated drill down is required.
        #  For the only table on the results page, find all <td> tags.  These table cells contain either
        #  label text for the type of weather data, or the data value as text.
        condensed_soup = soup.table.find_all('td')
        for index,item in enumerate(condensed_soup):
            #  Since the weather data value is always in the cell to the right of the data type label, when
            #  we find the desired data type we're looking for, enumerating the iterable allows us to use
            #  index + 1 to get the data value.
            if item.get_text() == 'Humidity':
                self.humidity = condensed_soup[index + 1].get_text().strip()

            if item.get_text() == 'Last update':
                self.last_update = condensed_soup[index + 1].get_text().strip()
```

weather_data function written for DataScrape/DataApp/views.py

```
def weather_data(request):

    # retrieve the current logged in user.
    user = request.user
    # get the user's data from the UserProfile model using the OneToOne user_id field.
    current_profile = get_object_or_404(UserProfile, user_id=user.id)
    # store the user's zipode in a variable
    zipcode = current_profile.zip_code

    #  WeatherScraper object is initialized with temperature, humidity, and last update
    #  time from the Weather.gov site page result obtained using the passed zipcode
    #  as a parameter to search for the local weather forecast.
    weather = WeatherScraper(zipcode)

    return render(request, 'DataApp/weather_data.html', {'weather': weather})
```

DataScrape/DataApp/templates/DataApp/weather_data.html written to display current local weather data.

```
{% extends 'base.html' %}

<!doctype html>
<html>
    <head>
        <title>{% block title %}| Weather {% endblock %}</title>
    </head>
    <body>
        {% block content %}
        <h2>Local Weather for Zip Code:  {{ weather.zipcode }}</h2>
        <h3>Temperature:  {{ weather.temp }}</h3>
        <h3>Humidity:  {{ weather.humidity }}</h3>
        <br>
        <h4>Last update as of:  {{ weather.last_update }}</h4>
        {% endblock %}
    </body>
</html>
```