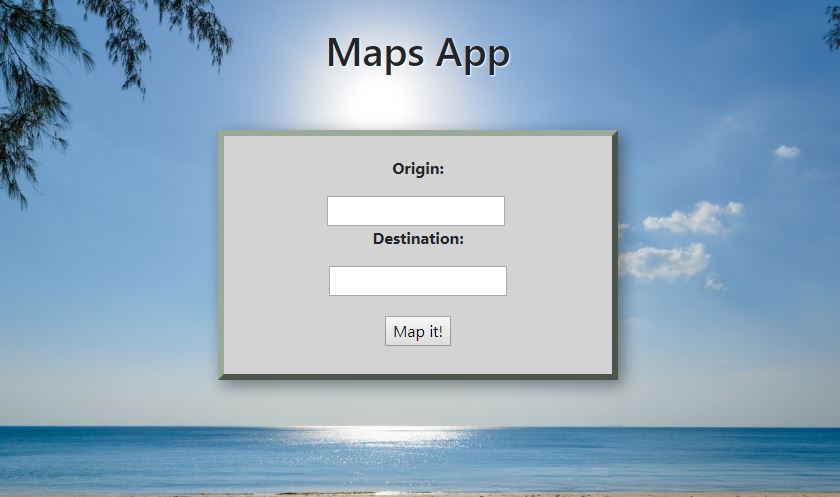
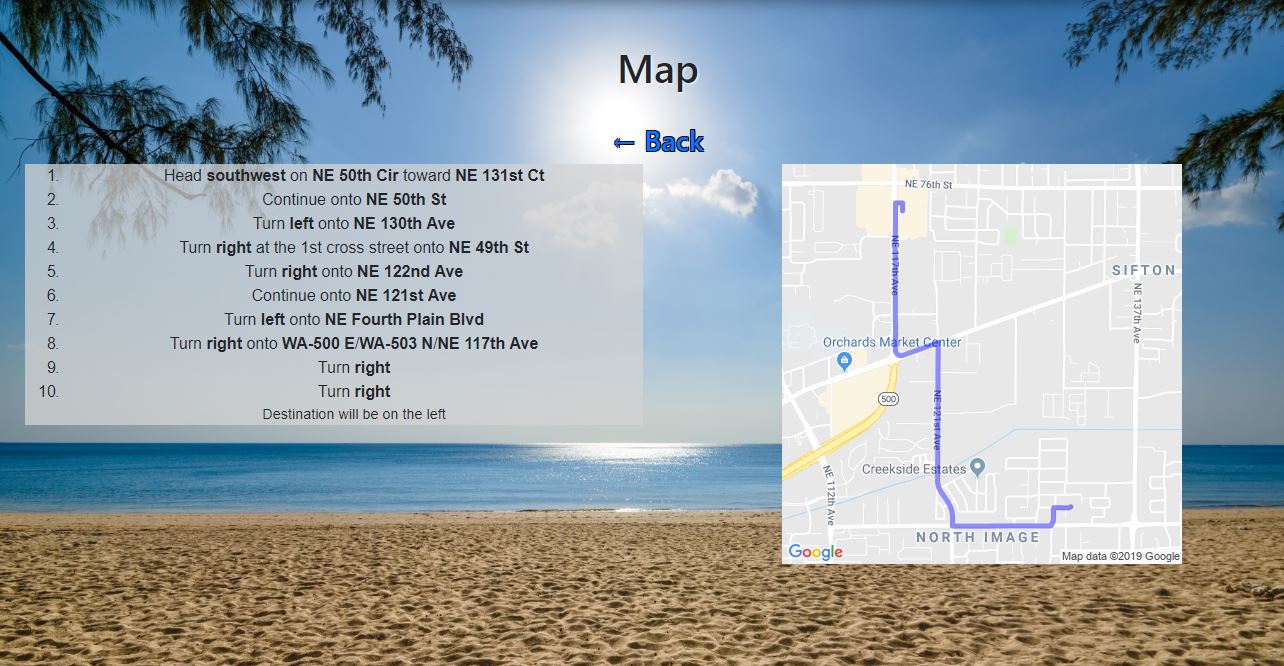
**Backend python code summary**

On this project for making travel helper app, I had my first successful attempt at working with django and creating an app using APIs that worked as intended. I enjoy working with Python as it is easy to use and you can get it to do anything you want. I like making stand alone programs with it but on this project I was tasked with making a web app with django. Django was a little difficult to use at first as it’s not like any other framework I’m familiar with, but it has the same basic structure of compartmentalizing the aspects of the project and having a flow between them. Once got an understanding of it I was off to the races to make my own modular app.

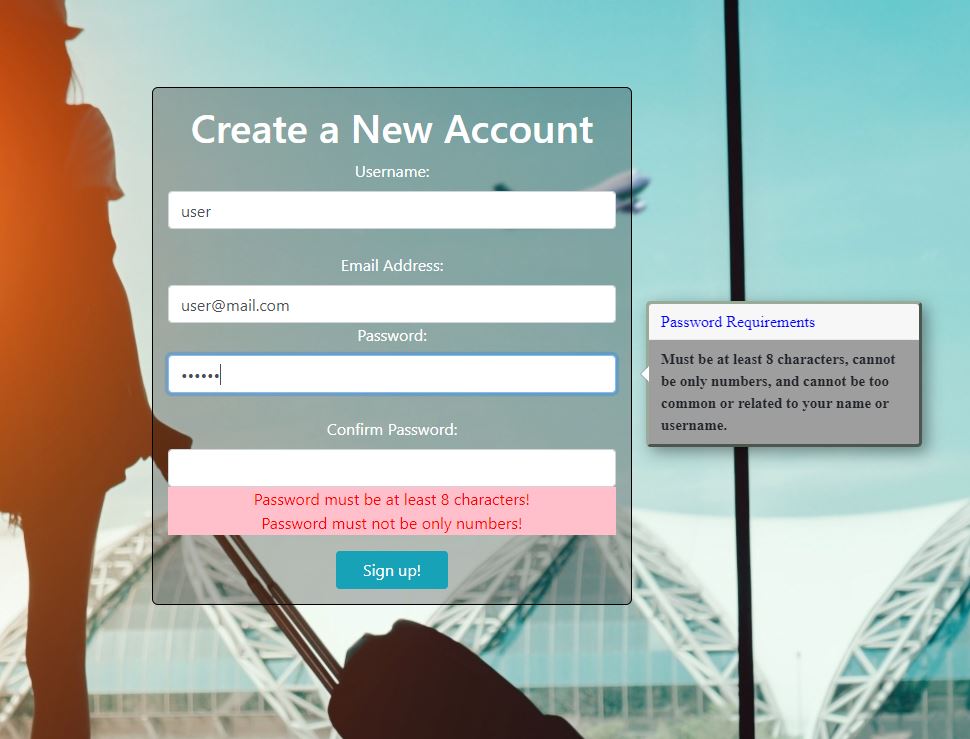
**Map App**

****

My story board was to create an app that could take in an origin and destination and then display the the direction steps in a list and show an image of the map.This was fun to work on because I got to work with the google maps APIs and there are about a million things you could do with it and I always like learning and working with APIs when it’s necessary. I used a Django form to take in the input from the user. When the data was validated and usable, it was then sanitized and formatted it into the API call. From that response was processed and opened with urllib and we end up with a bunch of JSON data. From that data the steps are parsed and added to a list to be printed in a for loop on the webpage. One thing is the steps are automatically formatted for HTML, it is convenient but can make styling a little trickier. Another thing that is returned from the first Directions API call is a polyline, which is a line that is drawn on the map showing all of the steps connected. The polyline is then fed into the Maps API call which returns a static image of the map with the directions drawn on, that can be displayed on the webpage.

|  |
| --- |
| def maps(request):  endpoint = 'https://maps.googleapis.com/maps/api/directions/json?'  map\_endpoint = 'https://maps.googleapis.com/maps/api/staticmap?size=400x400&path=enc:'  #api key for the directions api  api\_key = 'AIzaSyApD7f8-bekkFD9y57KfJIoP6qfkLLAKjk'  #api key for the static maps api, this api must have http restrictions, if you allow that on your directions api you only need one key  map\_key = 'AIzaSyBxvkmJqPkQxnWi3yzCkDQ8crUXtmVfbww'  if request.method == "POST":  form = MapsForm(request.POST)  if form.is\_valid():  origin = form.cleaned\_data['origin'].replace(' ', '+')  destination = form.cleaned\_data['destination'].replace(' ', '+')  #api call to directions api  nav\_req = nav\_req = 'origin={}&destination={}&key={}'.format(  origin, destination, api\_key)  api\_request = endpoint + nav\_req  response = urllib.request.urlopen(api\_request).read()  directions = json.loads(response)  #this parses the json response and sets the arrays we need to variables  routes = directions['routes']  legs = routes[0]['legs']  steps = legs[0]['steps']  step\_list = []  #this gets the directions steps and adds them to a list to be looped through on the template  for step in steps:  step\_list.append(step['html\_instructions'])  #this gets the line from the directions api response that is to be drawn on the static map  polyline = routes[0]['overview\_polyline']['points']  #api call to static maps  map\_url = '{}{}&key={}'.format(map\_endpoint, polyline, map\_key)  mapping = {  'origin': origin,  'destination': destination,  'directions': step\_list,  'map': map\_url,  }  context = {'mapping': mapping}  return render(request, 'MapsApp/mapsreturn.html', context)  else:  form = MapsForm(request.POST)  return render(request, 'MapsApp/maps.html', {'form': form}) |

**Front-end & Javascript**

****

I also did some Javascript work on this project for client side validation and added popovers in bootstrap.

var form = {

pword1: document.getElementById('pword1'),

pword2: document.getElementById('pword2')

}

form.pword1.addEventListener('change', function(e) {

if (e.target.value.length < 8 && isNaN(e.target.value) == false) document.getElementById('pwordResponse').innerHTML = "Password must be at least 8 characters!<br>Password must not be only numbers!";

else if (e.target.value.length >= 8 && isNaN(e.target.value) == false) document.getElementById('pwordResponse').innerHTML = "Password must not be only numbers!";

else if (e.target.value.length < 8 && isNaN(e.target.value) == true) document.getElementById('pwordResponse').innerHTML = "Password must be at least 8 characters!";

else document.getElementById('pwordResponse').innerHTML = "";

});

form.pword2.addEventListener('change', function(e) {

if (e.target.value != form.pword1.value) document.getElementById('pwordResponse').innerHTML = "Passwords do not match!";

else document.getElementById('pwordResponse').innerHTML = "";

});

This was just something added to help out the user to know if all of the inputs in the form are valid before they submit. Without this the user would have to just submit the form and hope it went through or have the page reset with the form cleared with an error message if there was a problem. The code makes comparisons to the account registration restrictions and the two passwords every time there is a change to one of the fields and shows an error at the bottom of the form before they even submit the entire thing. I could have made an error message list and appended every invalid input error but since my main job for the story was making sure the user knew if their password was valid this worked perfectly.