Zip Code Assessment

This assessment is designed to be written in two hours or less by an individual developer. It should take no special knowledge other than the things you've learned in this Angular course and your prerequisite knowledge of HTML, CSS, and JavaScript.

You are free to refer to your notes, the class materials. You are free to use the Internet to look things up but not to collaborate with another developer.

In this final assessment, your mission will be to create a new website from scratch that will allow your users to look up zip codes given a city and vice-versa.

Setting up the site and form

1. Create a new project called angular-project. Choose "no" for routing and "CSS" for styles.
2. Create a new component called CityLookup. Show this component when the project starts up.
3. Add a form to the component. It should have three <input>s, one each for "city", "state", and "country". Also give it a submit button.
4. When the user submits the form or clicks the submit button, your component should read in those three values and console.log them.
5. Next, add a <h3> below the form that says "Information for (city), (state)" where city and state are the actual values that the user entered in the form.
6. Make this <h3> only show after the user has submitted the form. (hint: create a property in the class called "haveData". When true, show the <h3>).
7. Bonus points for making it look nice with Bootstrap or any other CSS technique. But be careful to not spend too much time on cosmetics. Getting it functional through the last step is much more important.

Getting zip code data from the Internet

Your user can now enter a city, state, and country. Let's show them the data for that city. There is a service available called Zippopotam.us (http://Zippopotam.us) that allows you to send a GET request to http://api.zippopotam.us/country/state/city and get back a list of zipcodes for that city.

8. First, get familiar with the service by making a test request. Open your browser and type in http://api.zippopotam.us/us/md/columbia. Examine the returned JSON data.
9. Do the same for http://api.zippopotam.us/us/md/baltimore. See how they differ? Study the output for a minute. Maybe even keep this open for reference for future steps.
10. Instead of a simple console.log() when the form is submitted, make an Ajax call to Zippopotam.us to retrieve the data for that city.
11. Now display that data on the page below the <h3> you added above.
    - Hint 1: You'll want to iterate the data in the template and show it in s or maybe <tr>s.
    - Hint 2: Use interpolation in the iterator to put each data point in your or <td> or whatever you chose above
    - Hint 3: Zippopotam.us gives us a property unfortunately named "place name" (with the space in it). To interpolate this, use "*place['place name']"* instead of *"place.place name"*. Do the same with *"post code"* also. (Sheesh!)

Getting city data

12. Create a new component called ZipCodeLookup.
13. Make this the startup component.
14. It should have two <input>s for country and zip code. Also give it a button to submit the data. Feel free to copy/paste from your other page if you want to.
15. When the button is pressed, it should make a call to Zippopotam.us with the city and zip code like this http://api.zippopotam.us/country/zipcode. Again, read the response and display it on the page. (Hint: This endpoint, like the one above, returns an array so it should be iterated).
16. Run and test.

Setting up routing

Now that we have two components, let's make it a cohesive site by allowing navigation between them.

17. First, add a file called app.router.ts. (Hint: no need for ng generate. Just do it manually).
18. In that file create your routing array. (Hint: it's a JavaScript array of objects, each with a path and a component). Process it with RouterModule.forRoot().
19. Import this new configured router module into the AppModule. Also don't forget to import RouterModule.
20. Change your ZipCodeLookup component. The user should be able to click on the city name and navigate to the CityLookup component.
    - Hint 1: use a routerLink
    - Hint 2: They call it "place name" instead of "city"
21. Change your CityLookupComponent. When the user clicks on the zip code, it should navigate to the ZipCodeLookupComponent.
    - Hint: They call it "post code" instead of "zip"
22. Run and test. Make sure it is working before you go on.
23. Now change both of those links to include route parameters in the URL.
    - Clicking on the city name in ZipCodeLookup should also add the country, the state and the city to the url.
    - Clicking on the zip code in CityLookup should also add the country and the zip code in the url.
24. Again, run and test. Make sure they appear in the URL and navigation still works.
    - Hint 1: For navigation to work for these two new routes, you're going to have to add to your routing table.
    - Hint 2: The new routes will need route parameters
25. Change both components to read the route parameters from the url and pre-fill the textboxes that are already on the page.
    - Hint: you'll need to use dependency injection and put a couple of lines of code in either the constructor or ngOnInit
26. Once your textboxes are being filled from the route parameters, change the ngOnInit() method to check that the values are indeed being supplied and if so, go ahead and make the respective Ajax calls to fetch the data.

You'll know your site is working properly when you can fill in the data on either page, click the button to fetch data from the API server, then click on the results and navigate to the other page which then shows the proper data without you having to click any buttons.