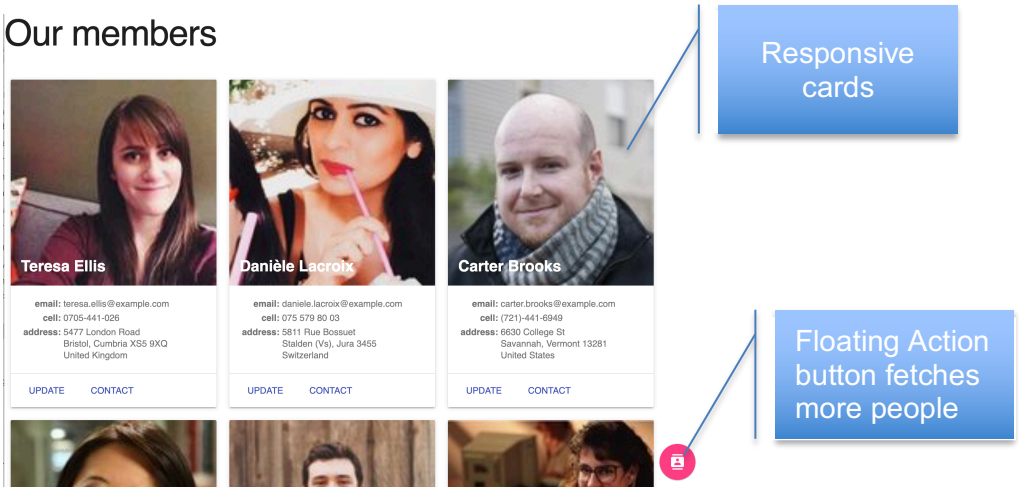
Ajax and Asynchrony

In this comprehensive lab we're going to build a screen that could look like this. →

With every click of the button in the lower right, we'll make an Ajax call to a RESTful API and get another batch of people to then load to the screen.



Setup

1. Copy the starters folder for ajax and take a look at the files in it. You have a people.html, a people.css, and a people.js file.
2. Install the libraries from package.json by running npm install.
3. Start a browser and open your people.html file just to make sure that it opens. Look at the console for any error messages.

You should see no errors and also no people yet. That's what we'll build up in this lab. Ready? Let's go!

Examining the API

4. Open a browser tab and navigate to https://randomuser.me/api/?results=5
5. Take a look at what is being returned. It may be helpful to open the browser's developer tools and study the network tab. If you notice, the value coming back is pure JSON. Remember that URL because we'll come back and use it in just a minute.

Crafting a call to the API

6. Open your index.js file and create a new method called getPeople().
7. Call getPeople() on two events. You can:
    a. The click of the getPeople button
    b. The document.DOMContentLoaded event.
8. Run and test, making sure that getPeople() is indeed being called.
9. In getPeople(), do a fetch from the URL we visited above.

Processing the API response

10. Remember from the lecture that fetch returns a promise. So, after your fetch call, do this:

```
.then(res => {
      const responseObject = res.json();
      console.log(responseObject);
      return responseObject;
```

```
    } )
```
That will convert the request string to a true JSON object.

11. Run and test again.
12. Take a look in the developer tools again and examine the data being returned. You should have an object that has our list of people inside it. What is the name of that key? ____
13. Chain another .then() function that will take the responseObject you saw in the step above and return only the property you discovered above. (Hint: you'll add one line of code and it will be another .then())
14. Run and test. If you've done it right, every time you refresh or click the button you'll see a new list of random people in the developer tools console.

Preparing to display the data

15. You'll need to do these things (in no particular order):

- Create an array to hold all of the people. Call it allPersonCards.
- Get a reference to the section with and id of 'people'. Call it peopleSection.
- Add a method called onePersonCard(person) that receives one person and returns a string that will be the HTML of that card. Just return a "Hello world" for now.

Displaying something when data is fetched

16. In the .then() methods after your fetch you have an array of people objects. Further process that array of persons into an array of personCards by calling onePersonCard(person). (Hint: Array.prototype.map())
17. **Then** process the array of personCards by adding them somehow to allPersonCards. (Hint #1: Array.prototype.unshift() or Array.prototype.push(). Hint #2: Don't forget about array destructuring).
18. **Then** set the peopleSection's innerHTML to all of the person cards. (Hint: Array.prototype.join())
19. That should do it! If you've gotten everything done, you should be able to refresh the page and see some number of "Hello world"s. If you hit the button, you'll see that list grow. Run and test.

Drawing the people

Very nice that we can see one "Hello world" for each person but wouldn't it be more fun to see a person?

20. Edit your onePersonCard method. Notice that it receives in a person. Also notice that each person object has sub-objects. For example, the *name* property is an object with a *title*, a *first*, and a *last*.
21. Use object destructuring to extract the person's first name.
22. Change "Hello word" to say "Hello <person's first name>". (Hint #1: Use backtics to create a template to display ${first}.)
23. Now do the same with last name, mailing address, cell, and email. Get the person's image from the *picture.large* property.
24. Put them all in the HTML.

Now this isn't a class on HTML and CSS so I've written all of that for you. Note that at the bottom of index.js, there's some commented out code to provide a nicely formatted template that will produce the screenshot that you saw at the start of this lab. Feel free to use that or create your own.

Refactoring

25. Change your getPersons method to receive in a "numberOfPeople" parameter with a default of 5. Use it in getPersons to fetch that number of people instead of the hardcoded number you have in there already.
26. Go through and pretty up your code, formatting it, tightening it, using good shortcuts wherever possible (like destructuring, spreading, object property shorthands, etc).

27. Bonus!! Process the fetch’s promise with async/await rather than with a .then().