Caching

In our previous lecture, we made calls to an API in our react elements. As you may have noticed, ever time that element loads, it makes that API call. While this is not necessarily a bad thing, if you know that the data you're calling is unlikely to change, it may be best to locally store that data in your UI's memory so that you're not making redundant API calls. There are a number of node modules which can create a cache. A general rule of thumb, though, is to go with the node module that has the greatest amount of support and downloads. For caching, one of the most downloaded node packages is: `node-cache` (though it does have fewer forks, it currently does have fewer active issues).

Installation:

To install node-cache in your project (and to make sure the dependency is added to your project so it can be run when downloaded by someone else), type:

```
npm i --save node-cache
```

Using caching:

When you cache something, you need to be relatively aware of how often the data you are caching is likely to change. If its something from a database that you know is almost never going to change, you can set your timeout for significantly longer than if you know your data will be different within a week (or day/hour/etc).

The basic structure of caching with node cache is:

- Initialization

```
// Importing the cache
import NodeCache from "node-cache";

// initializing the cache.
// Standard ttl is the default
// number of seconds for every cache
// object
const myCache = new NodeCache({ stdTTL: 100, checkperiod: 120 });

```

    - In your projects, try placing your instantiation of `NodeCache` inside of your components (functional). Instantiating the `NodeCache` inside of your component will create a new cache every time (so that you're not actually caching anything). In order to have persistence in your cache, you need to declare it outside of the component hierarchy.
- Retrieval

```
1  // It's good to use a constant as your key so that you always
2  // have the same key and don't accidentally use a mistyped key
3  const cacheKey = "aww";
4
5  // To get your stored items out of your cache, you simply
6  // need to use `.get(KEYNAME)`
7  const cachedStuff = myCache.get(cacheKey);
8
9  // When retrieving cached data, if the data have timed out,
10 // then they will be undefined. You can either use short
11 // circuiting, or use lodash/get, or deal with the logic elsewhere
12 const startingArrays = cachedStuff || [];
```

    - When retrieving from the cache, all you need to do is use the `.get` function. One thing to know, however, is that the `.get` function will return undefined if the cached element does not exist (or has timed out). With that being the case, it is always a good move to have a default value (like on line 12).
- Setting

```
1  // Here we set our cache with our key, then add in our desired
2  // data to be cahced (all posts), and finally enter our timeout.
3  myCache.set(cacheKey, allPosts, 100);
```

    - When setting the value in your cache, the parameters are `key`, `value`, and `time`. It's always a good move to use the key as a variable that's passed in so that you don't misspell when attempting to use a string literal.