AngularJS Capsules Lab

We know that any activities which we may be needed on multiple pages should probably be made a capsule and put under a shared folder. We did that with some services earlier. In this lab we'll get practice with making a factory.

Creating a cart factory

1. Create a new file under *shared* called cartFactory.js. In it create a factory that is part of the shared module. Remember that a factory returns a JSON object. Make sure yours does so.
2. Load that returned JSON object with an empty array called *cart* and these methods:
   - addToCart(product, quantity)
   - removeFromCart(product)
   - getCartTotal()

   These can be method stubs* for now that do nothing more than console.log() that they've been called.

Making the factory work in controllers

They're not doing much yet but let's see how they'd work on a page.

3. Edit productDetails.html and wire it up for Angular.
4. Change all the hardcoded values to Angular expressions like {{ product.productName }} and {{ product.unitPrice }}
5. Create productDetailController.js. Add a dependency to productService and one to your new cartFactory.
6. Hardcode a variable called productID to any number between 1 and 50. Just pick one.
7. Make a call to productService.getProduct() so you can get the product data from the RESTful service.
8. In the success callback, populate $scope.product.
9. Add an ng-click directive to the AddToCart button, calling cartFactory.addToCart(); Don't forget to pass in the product and quantity from the view.
10. Run and test. Make sure it is console.logging the right product and quantity.
11. Open checkoutController.js. Also make it depend on cartFactory.
12. When checkoutController loads, you're populating $scope.cart. Change that to do this:

```
$scope.cart = cartFactory.cart;
```

    This will allow us to tie the $scope's cart to the shared cart.

---

* A method stub is a function that exists, but doesn't really do anything. Developers use these as placeholders for real functionality later.

13. Reload and make sure it also is console.logging.
14. Find the <a> tag where we're removing from the cart. Make its ng-click action make a call to cartFactory.removeFromCart passing the product.

See how this is just the same as a service?

Persisting the cart

Well, that was fun, wasn't it? We've created the factory and we're calling factory methods. But the methods don't do anything but console.log. Let's fix that by making use of a RESTful api service.

15. Open cartFactory.js in your editor.
16. In the main cartFactory function, create a private reference to the factory itself like this:

```
var self = this;
self.cart = [];
```

17. Then check to see if there is anything in self.cart. If not, get it from the RESTful API like so:

```
if (! self.cart.length)
  $http({url: '/api/cart'}).then(function (res) {
    res.data.forEach(function (line) {
      self.cart.push(line);
})
  });
```

18. Lastly, locate where we're returning the object from the factory and instead of setting cart to an empty array, set it equal to self.cart.
19. addToCart(product, quantity) should make a POST request to /api/cart. It should pass a JSON object in the body with the product and the quantity. It should also push that product/quantity on to the self.cart array. It should return the promise. (Hint: a .then() also returns a promise).
20. removeFromCart(product) should make a -- you guessed it -- ajax DELETE call to /api/cart. It should pass a JSON object in the body with the product just like with addToCart. It also should remove that product from the self.cart array and return the promise. Note that since HTTP DELETEs don't normally have a body, you'll have to add this to your request:

```
headers: {"Content-Type": "application/json;charset=utf-8"}
```

21. Run and test the checkout and productDetail pages, adding and removing quantities from the cart. Keep adjusting until it works.

Bonus! Handling messages

22. You'll want to make adjustments to checkoutController.js to implement the callback to getCart(). (Hint: create a .then() and display a message of some kind). Same with the callback to removeFromCart(). Take care of both of those.
23. Do the same with productDetailController.js. When the addToCart() method is run, you may want to produce a message in a .then().