AngularJS Templates Lab

It makes sense to reuse code rather than duplicate it. HTML pages are no exception. We should do that with sections that are common to more than one page. Let's start with the footer.

1. Surf through your site. Notice that the header is exactly the same on every page. And, hey! The footers are, too. Repetition is not good. Let's fix that.
2. Edit your main page, index.html.
3. Go to the bottom of the page and find the footer. Cut it and replace it with this:

```
<footer class="row" ng-include="'/app/shared/pageFooterPartial.html'">
```

4. Now paste it into a new file called pageFooterPartial.html.
5. Run and test. You should still be able to see the page footer.
6. Now reuse this page footer in all other pages on your site by using the ng-include directive.

Replacing the header

You did it with the footer. Now let's do the same with the header.

7. Start with index.html. Find the bounds of the header and copy them into a new file called pageHeaderPartial.html.
8. Use ng-include to include pageHeaderPartial.html. But this time, also give it a controller called pageHeaderController.js.
9. The pageHeaderController doesn't need to do anything yet except console.log so you can see it running.
10. Do the same for all pages on your site and test them all out. You should see no change except the console.log() messages but now your pages are simpler.

Actions in a template

Now let's make the pageHeader do something much more useful. Let's show the user how many products they have in their cart at all times.

11. Open pageHeaderPartial.html. Notice that it has a link to the checkout page with a cart icon and a badge. That badge is supposed to have a number that shows how many products are in the cart. Right now it only has a hardcoded "0" in it.
12. Change that "0" to an expression with the value of cart.length.

If you run it now, it won't work because $scope.cart isn't a thing yet.

13. Give pageHeaderController a dependency on cartFactory. Remember that factories are singletons so if we set the cart length anywhere, it'll be reflected here as well.

14. At the top of the controller function, go
```
$scope.cart = cartFactory.cart;
```
15. That should do it! Go through the exercise of adding things to your cart and removing things from your cart. That badge will change as you change the cart contents.

Once all your pages are sharing a common header and footer, you can be finished.

Bonus!! Totalling the cart

You may have noticed that the cart has a subtotal but it isn't doing anything yet. We'll fix that in this bonus exercise.

The subtotal should be displayed in the view in an expression, populated in the controller based on the cart object, and calculated in the cartFactory. Let's do it!

16. Edit checkout.html. Find where the subtotal is being displayed and change it to an expression reflecting cart.subtotal.
17. Edit cartFactory.js. Add a method called calculateSubtotal() which will loop through the lines of the cart and sum up each unit price * quantity. Set self.subtotal to this value.
18. Each time we change the cart's contents this number should be recalculated. So call calculateSubtotal() in AddToCart(), removeFromCart(), and checkout().
19. Run and test this. You should see that the total always reflects the current contents of the cart.