AngularJS Routing Lab

In the SPAs lab we learned to create single-page apps. Let's expand that as we create route parameters and have a failover in case users request a bad address.

Getting a single product

1. Currently your route to the productDetail view is probably at "/". Make a copy of that route. On the copy, add a route parameter for '/:productID'. Don't forget the ":". This will allow us to scrape a productID from the URL.
2. Now let's read and use that parameter. Edit your productDetailController and add a dependency on $routeParams. $routeParams is a built-in provider that allows us to read any route parameters in our controllers.
3. Find where you're hardcoding the productID variable. Change it to read the productID from $routeParams.
4. Run and test. Adjust until you're able to see one product at the location http://<yourRoot>/app/product/index.html/#/XXXX where XXXX is a product id. Any valid product ID should work. Try several. Each should show a different picture, description, and all other product details.

Reordering the routes

5. Try to bring up the productList and the productSearch pages. Depending on where you have your new route, you'll either see these pages or they'll fail. If they've failed, see if you can fix the issue. (Hint: routes are evaluated top-down. The first matching route hit will be used and all others below it are ignored.)
6. Once that is fixed, you should also be able to click on any product in the productList and the productSearch pages and see the details for that product. Click on a few to make sure it works.
7. Bonus! The "/" route is now pointing to productDetail. Make it point to productList.

Creating a catch-all

8. Type a nonsense url into the address bar in your browser like http://<yourRoot>product/index.html/#/some/bogus/address. What do you see? _____
9. Instead, when the user types in nonsense, let's have him/her go to the product list. Add an otherwise route that sends the user to the productList.
10. Try your nonsense url again. In fact, try several. Make sure they all take you to the product list.

Getting a querystring

This last part has nothing to do with routing, but we really should know how to get querystrings. On our search page, we currently allow the user to type a searchString into the input box. We then use that searchString to limit our list via a filter. Let's improve on this. Let's allow the user to pass an initial searchString to our page via a querystring. Then we'll read that querystring in the controller and apply the filter immediately.

11. Edit the productSearchController. If you don't already have a dependency to $route, add one.
12. Read the queryString into the model like so:

```
$scope.searchString = $route.current.params.searchString;
```

    It should now be available to the view immediately when it is run.
13. Test it out by navigating to
    http://<yourRoot>/app/product/index.html/#/search?searchString=queso
    You should see only those products with 'queso' in the description.
14. Now, if you'll notice, in our pageHeaderPartial.html, the search form makes a GET request to "/search" when submitted. This means that it will tack on a searchString. So by reading $route.current.params.searchString above you just made the search box work on every page. To test that, go to any page on the site and type in a search term. Make sure that works.

Once it does, you can be finished.