

AngularJS Views Lab

Improving performance

We're currently using binding in several places. This is great except when we know that those elements will never change. Seriously, do we expect the company address to change during the run of the program?

1. Open the aboutus.html page.
2. Find all places where we are referencing the company address. Change every one to a one-time binding. (Hint: Use `{{ ::variable }}`).
3. Run and test.
4. Do the same on the index page with the featured products. Those may change, but not while the page is loaded in the browser.

Wiring up the checkout page

Let's work with the page that will be used to checkout the user. It'll have a review of their cart contents, their name, address, and a place for a credit card.

5. Find the *ordering* folder. The separate folder is hinting that it might be a good idea to create a separate module.
6. Create `orderingModule.js`. In it put the declaration for a new module called `orderingModule`. Don't forget your IIFE and the other best practices.
7. Add a `checkoutController.js` file. Create your new controller in it using the best practices we learned earlier.
8. In that controller, add a hardcoded JSON object called `cart` which might look like this:

```
$scope.cart = [
  {
    "product":{ /* Copy a product from mainProductController.js */},
    "quantity": 4
  },
  {
    "product":{ /* Copy a different product */},
    "quantity": 2
  },
  /* And so on ... */
];
```

Note: Look for a file called `cart.json.js` with a hardcoded cart in it. Using this file will save you from typing in some JSON data.

9. Go ahead and fill your cart contents with as many items as you'd like.
10. Underneath the controller but inside the IIFE, add this function:

```
function getCartTotal(cart) {
  // Loop through your $scope.cart array and return a sum
  // of all quantities * prices
};
```

11. In the controller, add these methods:

```

$scope.$watch('cart', function () {
    $scope.cartTotal = getCartTotal($scope.cart);
}, true);
$scope.removeFromCart = function (product) {
    // Remove the entire line from the cart for the product passed in
};
$scope.processOrder = function () {
    // console.log each item in the cart and then clear out the
    // cart contents
};

```

12. Add a method called `processOrder()`. It should `console.log` the cart details and then clear out all the cart lines. . This will simulate our order submission.

Displaying them in the view

13. In your `checkout.html` view, add links to the above needed scripts, to `bootstrap.css` and to Angular itself.
14. Find the `<main>` tag. Add an `ng-controller` directive pointing to your new `checkoutController`.
15. Display the cart contents in a table. (Hint: you'll use `ng-repeat` for each item in the cart.)
16. Add a row to your table to show the `cartTotal`.
17. Run and test. Make sure you can see your cart contents.
18. When the user hits the Place Order button, make sure `processOrder` runs. (Hint: `ng-click`).
19. Run and test the order submission. You should see the order details appear in the console and see the cart clear.

Altering cart items

The user may change his mind about an item. They may want to change the quantity of an item or delete it entirely.

20. Note the trash can icon to the right of each cart line. Add a click event to it that will call `removeLine` to remove just that line. (Hint: Angular keeps track of each line it adds, so you can just pass in *line.product* for the current product)
21. Since you already wrote `removeLine()`, you should be able to test it out. Click the trashcan icon and see if that line is deleted.
22. In your `ng-repeat`, find where you've bound quantity. Replace that expression with something like this:

```
<input type="number" ng-model="line.quantity" />
```

 where *line* is your range variable.
23. Run and test. This should work without *any* extra coding! Angular rocks!

Once you can submit your order and remove items from your cart, you can be finished.