Directives set the stage in your html page

Controllers give you r wrap the behavior that it lacks

Write expression so that you can add your data to the View.

Modules make connections with dependency injections

4.2 Refactoring Description Tab 250 PTS

Notice that we have created an empty html file called `product-description.html`. Separate out the `Description` Tab's content into the new html file. Include the `product-description.html` in our index where it belongs.

Help Me

Task 1/2

Separate out our description tab into `product-description.html`.

index.html | product-description.html | app.js

```
<!-- Move the description tab here! -->
<h4>Description</h4>
        <blockquote>{{product.description}}</blockquote>
```

Task 2/2

Include `product-description.html` inside the description tab.

```
<div ng-show="tab.isSet(1)"
     ng-include="'product-description.html'">

</div>
```

4.3 Creating an Element Directive 250 PTS

Instead of using ng-include, lets create a custom directive called `productDescription`.

Help Me

Task 1/2

Create an Element Directive for our `Description` div that includes the `product-description.html`.

```
app.directive("productDescription", function(){
  return{
    restrict: "E",
    templateUrl: "product-description.html"
  };
});
```

Task 2/2

Include the file on the page using a custom directive tag instead of `ng-include`.

```
<div >
    <product-description ng-show="tab.isSet(1)"></product-description>
</div>
```

4.4 Creating an Attribute Directive 250 PTS

As you probably have noticed, we have built out more information on the `specs` tab. Let's refactor the contents of our specs div into an attribute directive.

Help Me

Task 1/3

Move the contents inside the `specs` div to the `product-specs.html`.

index.html | product-specs.... | app.js | product-

```
<h4>Specs</h4>
<ul class="list-unstyled">
  <li>
    <strong>Shine</strong>
    : {{product.shine}}</li>
  <li>
    <strong>Faces</strong>
    : {{product.faces}}</li>
  <li>
    <strong>Rarity</strong>
    : {{product.rarity}}</li>
  <li>
    <strong>Color</strong>
    : {{product.color}}</li>
</ul>
```

Task 2/3

Create a new attribute directive for our specs tag called `productSpecs`. Have it use our new `product-specs.html` template.

```
app.directive("productSpecs", function(){
  return{
    restrict: "A",
    templateUrl: "product-specs.html"
  };
});
```

Task 3/3

In `index.html`, use your new attribute directive to show the product specs.

```
<!--  Spec Tab's Content  -->
<div product-specs ng-show="tab.isSet(2)">

</product-specs></div>
```

4.5. slides

4.6 Refactoring Product Tabs 250 PTS

We feel like the Product Tabs section could be better managed if it were a directive.

Help Me

Task 1/7

Create an **element** directive called `productTabs`.

Task 2/7

Tell your new directive to use the `product-tabs.html` template with the `templateUrl` attribute.

```
app.directive("productTabs", function(){
  return{
    restrict: "E",
    templateUrl: "product-tabs.html"
  };
});
```

Task 3/7

Give our `productTabs` directive all the tab functionality that is currently inside our `TabController`. Make sure that you **do not** delete the `TabController` yet. We don't want to break the site.

```
app.directive("productTabs", function(){
  return{
    restrict: "E",
    templateUrl: "product-tabs.html",
    controller:function() {
      this.tab = 1;

      this.isSet = function(checkTab) {
        return this.tab === checkTab;
      };

    this.setTab = function(setTab) {
    this.tab = setTab;
    };
  },
  };
});
```

Task 4/7

Add the `controllerAs` attribute to your directive setting it to `tab` so the directive knows what all the references to tab in `product-tabs.html` are.

```
  this.setTab = function(setTab) {
  this.tab = setTab;
  };
},
  controllerAs: "tab"
```

Task 5/7

Put the tabs `section` inside of the provided `product-tabs.html`. Remove the `ng-controller` from the `section` inside the file once moved.

app.js | index.html | product-ta... | produc... | produc... | produc...

```
<section >
  <ul class="nav nav-pills">
    <li ng-class="{ active:tab.isSet(1) }">
      <a href ng-click="tab.setTab(1)">Description</a>
    </li>
    <li ng-class="{ active:tab.isSet(2) }">
      <a href ng-click="tab.setTab(2)">Specs</a>
    </li>
    <li ng-class="{ active:tab.isSet(3) }">
      <a href ng-click="tab.setTab(3)">Reviews</a>
    </li>
  </ul>

  <!--  Description Tab's Content  -->
  <div ng-show="tab.isSet(1)" ng-include="'product-descriptio
  </div>

  <!--  Spec Tab's Content  -->
  <div product-specs ng-show="tab.isSet(2)"></div>

  <!--  Review Tab's Content  -->
  <product-reviews ng-show="tab.isSet(3)"></product-reviews>

</section>
```

Task 6/7

Now remove the product tabs `section` from `index.html` and the `TabController` from `app.js`.

Task 7/7

Use our new `productTabs` directive where the tabs `section` used to be in our `index.html`.

```
    <!-- Product Tabs  -->
     <div><product-tabs></product-tabs></div>
  </div>
```

4.7 Refactoring Product Gallery 250 PTS

Now that we've separated the Product Tabs, why not separate the Gallery too?

Help Me

Task 1/7

Create an **element** directive called `productGallery`.

Task 2/7

Tell your new directive to use the `product-gallery.html` template with the `templateUrl` attribute.

```
app.directive("productGallery", function(){
  return{
    restrict: "E",
    templateUrl:"product-gallery.html"
  };
});
```

Task 3/7

Give our `productGallery` directive all the gallery functionality that is currently inside our `GalleryController`. Make sure that you **do not** delete the `GalleryController` yet. We don't want to break the site.

```
app.directive("productGallery", function(){
  return{
    restrict: "E",
    templateUrl:"product-gallery.html",
    controller: function(){
      this.current = 0;
      this.setCurrent = function(imageNumber){
      this.current = imageNumber || 0;
        };
      }
  };
});
```

Task 4/7

Add the `controllerAs` attribute to your directive setting it to `gallery` so the directive knows what all the references to gallery in `product-gallery.html` are.

```
      this.setCurrent = function(im
      this.current = imageNumber ||
        };
      },
    controllerAs: "gallery"
  };
```

Task 5/7

Put the gallery `div` inside of the provided `product-gallery.html`. Remove the `ng-controller` from the `div` inside the file once moved.

app.js | index.ht... | product... | pro... | pro... | pro... | pro...

```
<div ng-show="product.images.length">
  <div class="img-wrap">
    <img ng-src="{{product.images[gallery.current]}}" />
  </div>
  <ul class="img-thumbnails clearfix">
    <li class="small-image pull-left thumbnail"
        ng-repeat="image in product.images">
      <img ng-src="{{image}}" />
    </li>
  </ul>
</div>
```

Task 6/7

Now remove the image gallery `div` from `index.html` and the `GalleryController` from `app.js`.

Task 7/7

Use our new `productGallery` directive where the gallery `div` used to be in our `index.html`.

app.js | index.html | product-gallery.html | Toggle Navigation | product-descripti... | product-specs.htm

```
<!DOCTYPE html>
<html ng-app="gemStore">
  <head>↔</head>

  <body ng-controller="StoreController as store">
    <!--  Store Header  -->
    <header>↔</header>

    <!--  Products Container  -->
    <div class="list-group">
      <!--  Product Container  -->
      <div class="list-group-item" ng-repeat="product in store.products">
        <h3>{{product.name}} <em class="pull-right">{{product.price | currency}}</em></h3>

        <!-- Image Gallery  -->
      <div><product-gallery></product-gallery></div>

        <!-- Product Tabs  -->
        <product-tabs></product-tabs>

      </div>
    </div>
  </body>
</html>
```

Congratulations!

You've completed Level 4 of **Shaping up with AngularJS**.

Share on: