# Forms Intro Lab

## Angularizing the login and register pages

1. Look in the auth folder. You'll see two pages, login.html and register.html.
2. In this folder, add an authModule.js file which defines the authModule.
3. Add <script> tags to the tops of login.html and register.html to your new authModule.
4. Add ng-app="authModule" to the <body> tag.
5. Add two new controllers called loginController.js and registerController.js. They should both be created using our best practices and should both have $scope injected.
6. Open both HTML files and find the <main> tag in each. Put ng-controller attributes on them so Angular will use our controllers. Don't forget the <script> tags to include your respective controllers.

## Wiring up the login page

Now that angular is in charge of these pages, let's bind each to its model.

7. Add this to your loginController.

```
$scope.login = function (user) {
  $scope.successMessage = "You are now logged in as " + user;
};
```

8. Open login.html. Note that there's a <p class="alert alert-success"> at the top. Give that a one-way binding to successMessage. Make it show only when successMessage is set. (hint: ng-show)
9. Find the <form>. Give it a name. Remove action and method. Make it call your login function above when the user submits the form.
10. Find the inputs. Bind each to a model variable.
11. Run and test. When you click the button you should be able to see the username in the login message.

## Wiring up the registration page

12. Add a method to your registrationController called $scope.register which is similar to your login method from above. It should set a successMessage.
13. The success message is at the bottom of register.html this time. Bind it to successMessage $scope variable.
14. Wire up the form submit to $scope.register(), removing action and method.
15. Connect all inputs to $scope models.
16. Feel free to put testing values in as mustaches. Run and test. Remove the testing mustaches before moving on.

Now let's tame our checkout page! Look in the *ordering* folder for all of the below.

## Wiring up the shipping costs

17. Add a new controller called shipViaController as part of the orderingModule.
18. Add this to the controller:

```
$scope.shipViaOptions = [
  {id: 1, name:'Next day', price:100},
  {id: 2, name:'Two day', price:50},
  {id: 3, name:'Ground', price:0}
];
```

19. Open checkout.html and make sure that controller file is included in a <script> tag.
20. Find the <section> where the checkout page offers shipping options. Add an ng-controller directive to it pointing to your new shipViaController.
21. Populate that dropdown using ng-options. The shipVia name should be displayed in the box.
22. Don't forget to add an ng-model='shipVia' to bind the option chosen at run time.
23. Add this temporary test expression anywhere in the section:

```
{{ shipVia }}
```

24. Run and test. Once you're convinced that it is working, remove your test expression.
25. Bonus!! See if you can make the name and the price show up in the dropdown. (Hint: use string concatenation with the "+" sign in ng-options).

## Wiring up the ship-to section

Now let's do the same with the ship-to section.

26. Add a new controller called shipAddressController.js.
27. Add a fake logged-in customer to it:

```
$scope.customer = {
  "companyName": "Bottom-Dollar Markets",
  "contactName": "Elizabeth Lincoln",
  "address": "23 Tsawassen Blvd.",
  "city": "Tsawassen",
  "region": "BC",
  "postalCode": "T2F 8M4",
  "country": "Canada"
};
```

28. Add an ng-controller directive to the section tag pointing to your controller.
29. Find the <form> tag in that section and remove the action and method attributes.
30. Add a name attribute to the form and make sure each <input> has a name and an ng-model so that the customer shows up in the form.
31. Run and test. Feel free to put some dummy mustaches on the page to debug your values but if you do, don't forget to remove them.

Once you have well-behaving forms, you can be finished.