Angular CLI Lab

Let's create the basis for our Angular-ized WMS. We'll start off by creating the bare-bones Angular app and then add some components to it.

Creating a starter site

1. Open a command window and cd to the root of your project.

```
ng new warehouse
```

   If it asks you about settings, answer like this:
   - Would you like to add Angular routing? - No
   - Which stylesheet format would you like to use? - CSS

   Note: this command may take a while to run mostly because of the installation of the supporting libraries. And it may complain about not knowing who you are on github, Both of these are normal.
2. Notice that there is now a new directory called 'warehouse'. cd to it. Look around at the files created.

Using the development server

Let's use the built-in server that comes with the Angular CLI.

3. Do this:

```
ng serve
```

4. Look at the message. Make sure it says that it is running a server at a particular port.
5. Point your browser to localhost at that port. Make sure your page comes up. You should see Angular's default page in the browser with "warehouse" near the top.
6. Keeping your browser open, edit the app.component.ts file in your favorite IDE.
7. Change the *title* variable to "Northwind Traders Warehouse Management". Hit save and watch the browser window. You should see your application reload without refreshing. This is the file watcher in action.
8. Go ahead and select everything in app.component.html and replace it with this:

```
<h1>
  {{title}}
</h1>
```

9. Once again, when you save it, you should see your new title appear in the browser but you've gotten rid of Angular's boilerplate content.

Working with angular.json

The Angular CLI makes some decisions for us that we may not agree with. We can change some of those by changing the config file called angular.json. Let's say we want to put our images in a different location than the default demands.

10. Delete favicon.ico in the src folder.
11. Find the assets folder and create two new directories under it. Create "css" and "js"
12. Look in the setup/assets folder. Do you see the images folder? Move it and all of the contents into the assets folder you just created. You should have a directory called warehouse/src/assets/images and in there you should see favicon.ico along with some other images.
13. Edit angular.json. Find the line where the favicon is included in the build. Remove that line and make sure the JSON is valid.
14. Quit ng serve and run it again. Adjust angular.json until it validates.

15. Open index.html in your IDE and change the favicon line in the head to the proper folder (assets/images/favicon.ico). Keep adjusting until you see your new favicon.
16. Back in angular.json, look for the *prefix* key. It currently says "app". Change it to something related to this project. Suggestion: You could use "nw" which is short for "Northwind Traders", the name of our company. That's what we'll be using in these labs.

Adding sitewide styles

17. We've decided to use Bootstrap for styling. Install jQuery and Bootstrap like so:

```
npm install jquery bootstrap@^3
```

18. This will have created a new folder under node_modules. See if you can find it. You'll need that in the next few steps.
19. Open the angular.json file. Find the */projects/warehouse/architect/build/options/styles* key. Notice that it is an array. Add Bootstrap's css file to that array as a string. (Hint: It might be located in something like "node_modules/bootstrap/dist/css/bootstrap.min.css")
20. Find the */projects/warehouse/architect/build/options/scripts* key. Did you find it? In there add jQuery (Hint: try "node_modules/jquery/dist/jquery.min.js"). Then do the same with Bootstrap's JavaScript. (Hint: maybe "node_modules/bootstrap/dist/js/bootstrap.min.js"?).
21. Stop and restart the development server so your changed angular.json file will be read.

If you see the font change to a sans-serif font, you've got it working properly.

Scaffolding components

22. Using the Angular CLI, create these four components (HInt: don't forget about the --dry-run option if you want to see what is about to happen).
    - Dashboard in a 'dashboard' directory
    - Inventory in a 'inventory' directory
    - ReceiveProduct in a 'receiving' directory
    - ShipOrder in a 'shipping' directory
    - OrdersToShip in that same 'shipping' directory

Hint: to make the last three, you'll need to do something like

```
ng generate component --flat shipping/shipOrder
```

23. Look in the three folders and examine the files created by this. You should see four new files for each component created.
24. Make the receiveProductComponent appear on the page by editing app.component.html and adding this to it

```
<nw-receive-product></nw-receive-product>
```

25. Run and test. When you see your new component appear on the main page, you know you did it right.

Giving the components a little shape

We've created some components but you have to admit, they're all pretty boring, right? Let's give them some content. But we know you'd rather focus on Angular instead of HTML so rather than you taking a lot of time to write these components by hand, we'll give you starters.

26. Look for a folder called setup/assets/html. And in that folder you'll find some html files you can use for your components. They're full of hardcoded, fake data.
27. Edit receive-product.component.html. Replace its content with the receive-product HTML file in the setup directory.
28. Do the same for orders-to-ship.component.htm, ship-order.component.html and dashboard.component.html.
29. Give each of those a look by changing app.component.html so each gets a turn at being the startup component.

Once you've seen the four components, you can be finished.