Observables Lab

In this lab we're going to make the ShipOrderComponent work with real data instead of the hardcoded values we've had so far.

Getting locations for those orders

Remember where we're hardcoding a location from which to pick? Let's get an actual location from the database.

1. Open ship-order-component.ts. Find your getBestLocation method where you hardcoded the return.
2. Change that hardcode to hit the GET endpoint for /api/locations/forProduct/:productID. Of course you're passing the productID in that productID parameter. Make sure you're subscribing a function. In that success callback, set your location.
3. Run and test. When the button is pressed by the user, a real location should be displayed. You'll know you got it right when there's a different location for each product.

Set the shipped or problem flag

4. Still in ship-order-component, find the method you're running when the "Mark as problem" checkbox is checked.
5. Make that send a PATCH request to /api/orders/<orderID>/MarkAsProblem. (Note: the PATCH protocol requires a body but our service ignores it, so send an object -- even an emply one).
6. Do the same for "Mark as shipped"
7. Run and test them both. When you mark as shipped, you shoud see that the inventory amount is reduced by the amount shipped.

Remember, you can always look at an order and manually set the order status by using mongo directly. Here's a sample for setting the status of order 10 back to 0:

```
db.orders.update( {orderID:10}, {$set: {status:0} } );
```

Receive product

Remember how when we receive product we're displaying the productID and quantity? We'd like to display an entire product on the page instead of merely the productID. Let's go grab a product from our server.

8. As soon as the user enters a productID, make an Ajax GET call to /api/products/<productID>. The response will contain the details for that productID. On success, populate this.product from the response so it will display your product on the form. On failure, show a message that that productID is not found.

Services Lab

We haven't implemented security yet but in preparation, let's set up a few features. Let's say that once the user is logged in we want to display his/her name on each page. What are some ways this could be done?

Obviously we don't want to have the user log in on every component. We could pass the user's identity around using property binding, but that's a whole lot of properties. We'd have to do it in every component.

This is where services shine! If we were to inject that service in every component and set a user propery in a login component, it could be seen in all of them. Let's work on making that happen.

1. Create a new component called Login. In the class, add string properties for username and password.
2. In the template, create:
    - inputs for username and password,
    - a button to log them in. The button's click event should run a method called login().
    - a to hold a success/failure message for their attemped login.
3. The login method will pretend to log them in and create a new user object. This object should hold whatever information you want it to, but at minimum it should have a userid, username, password, givenName, and familyName. Set these values to whatever you like in the login method. You can use an Ajax call to get a user from /api/customers if you want to.
4. Now let's use the login component. Add a route to it in app.router.ts. Then add a router link to it in the App component.
5. Run and test. Make sure that you can get to the component and that 'logging in' (wink wink) will show a success message and set the user object's properties.
6. While you're still running, navigate to the Receive Product component or the Orders Ready to Ship component.

Our goal is to make the user data appear in those other components.

Creating the service

7. In the Angular CLI, create a new service called login service. It should probably live in the shared folder. Something similar to this might do the trick:

```
ng generate service shared/Login --module=app.module
```

8. Give that service a property called user. It doesn't need to have any properties here unless you want it to.

Note that this is a one-liner service so far. Super-simple.

Using the service

9. Inject the service into your login component. After the user has logged in, set this service's user property equal to the user entity you created in the component.
10. Now inject the service into your receive product component.
11. In the ngOnInit, set a private user property to the service's user property.
12. Go into the template and use interpolation to disply the user's given name and family name.
13. Run and test. Login and then navigate to the receiving component. Do you see your name? Cool! You've just shared data through a service!