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.

Real data in the order

1. Run your application. Look at either the dashboard or the orders to ship component. Click on an order. You should be sent to http://localhost:4200/ship/<orderId>. Then look at the orderId in the url and on the page. Do they match? ______ They should. Now look at the order date, the ship_via, the shipping label at the bottom and the list of products. All of those are hardcoded so they will always say the same thing. Let's get some real data from our Ajax API.
2. Open ship-order.component.ts. Prepare it to make Ajax calls by importing and injecting HttpClient. (Hint: This is what you did in the last lab and you should be familiar with it by now).

In ngOnInit, you're properly getting the orderId from the URL. But then you create fake hardcoded data that looks like an order.

3. Delete the code where you're creating that fake order.
4. Change ngOnInit to populate this.order from the Ajax API. You'll need to ...
    - import and inject HttpClient
    - import the map function from rxjs/operators
    - Make a get call to /api/orders/<orderId>
    - Pipe it through the map function to convert the raw response to an actual Order
    - subscribe to the observable with a function that sets this.order = order.
5. Run and test. Make adjustments until you can see all the right order details on the page. You should see the right order date, ship via, shipping address, the right products in the list, and the right shipping address in the shipping label at the bottom of the component.

Getting locations for those products

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

6. Still in ship-order.component.ts, find the getBestLocation method where you hardcoded the return.
7. Change that hardcode to hit the GET endpoint for /api/locations/forProduct/:productID. Of course you're passing the productID in that productID parameter.
8. Note that what comes back from the API call is an array of Locations so you'll want to convert the response to Array<Location> using the map function.
9. Make sure you're subscribing a function. In that success callback, set your orderLine's locationID.
10. 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

11. Still in ship-order.component, find the method you're running when the "Mark as problem" button is clicked.
12. 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). We're doing nothing with the response so no need to map anything. Don't forget to subscribe() or the observable never gets run.

13. Run and test. Mark one as problem. Then look at your list of orders and you'll see that it is no longer ready to be picked because you've said it has a problem. When you can mark it as problem and see it removed from the list, you know you've got it working.
14. Do the same for "Mark as shipped"
15. Run and test them both. When you mark as shipped, you should 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.

16. 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.