COMP3322 Modern Technologies on World Wide Web

Lab 7: RESTful Web Service Using the Mean Stack

Introduction

In this lab exercise, we will re-implement the client-side of the RESTful Web service in lab6 using AngularJS. Similar to lab 6, the web service allows retrieving, adding, updating and deleting commodities from a MongoDB database. The client-side AngularJS application provides the web page for displaying commodities, adding, updating and deleting them. In this way, the web application is implemented with the MEAN (MongoDB, Express.js, AngularJS and Node.js) stack.

Set Up the Express Project

We will reuse the express server app implemented in Lab 6. Therefore, make a copy of your "lab6" project folder (which you created when working on Lab 6) and name it "lab7". We will reuse "**package.json**" from Lab 6 in which you have added dependencies for MongoDB, the modules you have installed for the express server app (in "./node_modules" folder), and the MongoDB database you created in Lab 6 (the database file are stored under ./data folder).

Open **app.js** and comment out the following lines of code, since we do not need the pug view engine in this project. You can also remove the ./views folder.

```
app.set('views',path.join(_dirname,'views'));
app.set('views engine', 'pug');
```

Instead, we will implement the client-side code in **index.html** and **externalJS.js** which will be placed under ./public and ./public/javascripts, respectively, and accessed as static files. Therefore, comment out the following lines of code in **app.js** as well. You can also remove **index.js** from the ./routes folder.

```
var routes = require('./routes/index');
app.use('/', routes);
```

All other lines of code in **app.js** can remain, including those for loading MongoDB related modules. Note that since we are reusing the database created in Lab6, the database in the code should still be 'localhost:27017/lab6'.

In addition, we will use the same router ./routes/users.js which you implemented in Lab 6 as the back-end in this web service application.

Launch the MongoDB Database

We will follow the same steps as done in Lab 6 to launch the database server. Launch a terminal and switch to the directory where MongoDB is installed. Start MongoDB server using the "data" directory of "lab7" project as the database location, as follows: (replace "**YourPath**" by the actual path on your computer that leads to "lab7" directory)

If you use a 64-bit MongoDB on your own computer, please use the following command:

```
./bin/mongod  --dbpath YourPath/lab7/data
```

If you use a 32-bit MongoDB, please use the following command instead:

```
./bin/mongod  --storageEngine=mmapv1 --dbpath YourPath/lab7/data
```

After starting the database server successfully, you should see some prompt in the terminal like "I NETWORK [initandlisten] waiting for connections on port 27017". This means that the database server is up running now and listening on the default port 27017. **Then leave this terminal open and do not close it during your entire lab practice session,** in order to allow connections to the database from your Express app.

Launch another terminal and switch to the directory where mongodb is installed. Then you can use the following commands to add commodities into the database, if you need them for testing purpose. (If there are still commodities in your "lab6" database as you inserted last time, you do not have to do this step.)

```
./bin/mongo
use lab6
db.commodities.insert({'category':'Computer', 'name':'lenovo', 'status':'in stock'})
```

Lab Exercise: Create the AngularJS web client-side

Task 1: Download lab7-materials.zip from Moodle, unzip it and you will see two files: **index.html** and **externalJS.js**. Copy **index.html** to ./public/ folder and **externalJS.js** to ./public/javascripts/ folder. The given **index.html** contains HTML content of the page. We will implement AngularJS code in both files.

Like in Lab 6, you can always check out the web page as follows. Launch a terminal, switch to your "lab7" directory, and type "**npm start**" to start the Express app (**you should always use control+C to kill an already running app before you start the app again after making modifications**). Then check out the web page at http://localhost:3000/index.html on your browser. You should see a page like the following:

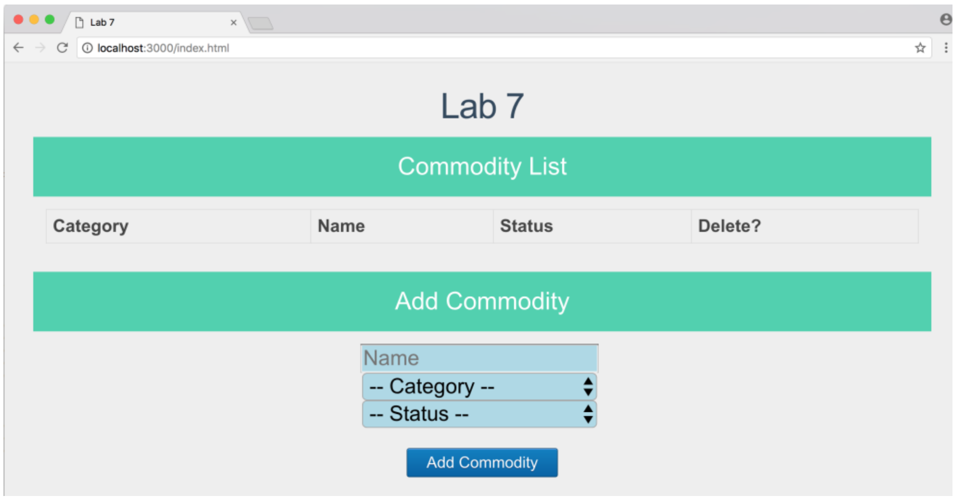


Fig. 1

Task 2: Open **index.html** and add the following code into the <head> element, which loads the AngularJS framework.

```
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.4/angular.min.js"></script>
```

Then add the following attributes into the <body> tag. Here directives ng-app and ng-controller define the AngularJS application and a controller, respectively.

```
ng-app="commodityListApp" ng-controller="commodityListController"
```

Open **externalJS.js**. You will see there are codes to create an AngularJS module (i.e., AngularJS application “commodityListApp”) and add the controller “commodityListController” into the application.

Task 3: We now add codes for displaying the commodity list in **externalJS.js**. Add the following code into the constructor of the “commodityListController” controller (where we mark “to be completed”).

```
$scope.commodities = null;
$scope.updateStatusOrNot={};
$scope.new_status={};

$scope.getCommodities = function(){
      $http.get("/users/commodities").then(function(response){
         $scope.commodities = response.data;

           for (var i=0; i<$scope.commodities.length; i++){
                   var commodity = ($scope.commodities)[i];
                   $scope.updateStatusOrNot[commodity._id]=false;
                   $scope.new_status[commodity._id]="";
           }
      }, function(response){
         alert("Error getting commodity:"+response.statusText);
      });
   };
```

The function getCommodities() sends an AJAX HTTP GET request for http://localhost:3000/users/commodities . Recall that on the server side, the end point is handled by the following middleware in ./routes/users.js (which we implemented during Lab 6):

```
/*
* GET CommodityList.
*/
router.get('/commodities', function(req, res) {
           var db = req.db;
           var collection = db.get('commodities'); collection.find({},{},function(err,docs){
                   if (err === null)
                           res.json(docs);
                   else res.send({msg: err });
           });
});
```

After receiving server response, getCommodities() stores the received JSON object in variable commodities (in the $scope object), and add an entry for each commodity in updateStatusOrNot and new_status, respectively, which can be used in the AngularJS code in **index.html**.

Add the following directive as the last attribute in the <body> tag in **index.html**. ng-init executes some code when the AngularJS application is initialized (http://www.w3schools.com/angular/ng_ng-init.asp).

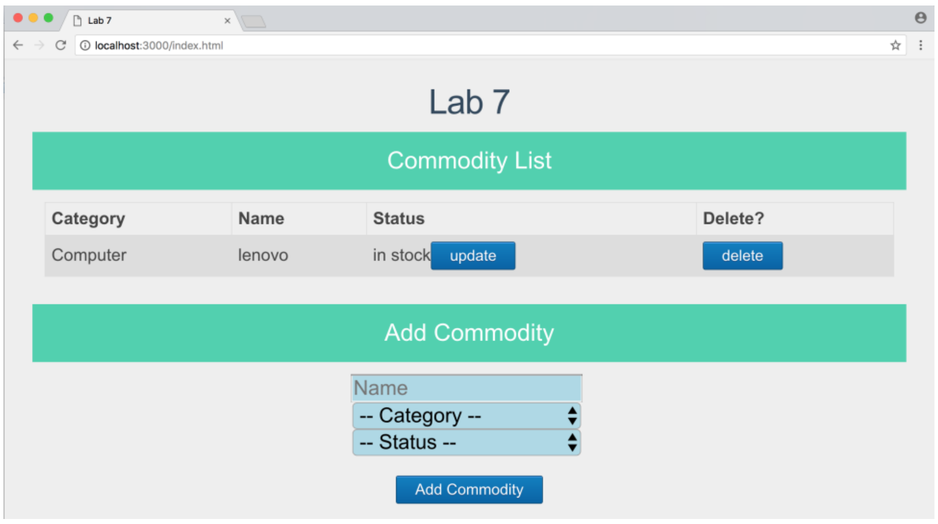
```
ng-init="getCommodities()"
```

Then add the following code into <table id="commodityList"> </table> element in **index.html**.

```
 <tr ng-repeat='commodity in commodities'>
            <td>{{commodity.category}}</td>
            <td>{{commodity.name}}</td>
            <td id={{'status_'+commodity._id}}>
                 <span ng-hide="updateStatusOrNot[commodity._id]">{{commodity.status}}<button
class="myButton" ng-click="showStatusOptions(commodity._id)">update</button></span>
                 <span ng-show="updateStatusOrNot[commodity._id]">
                      <select ng-model="new_status[commodity._id]">
                              <option value="">-- Status --</option>
                              <option value="in stock">in stock</option>
                              <option value="out of stock">out of stock</option>
                      </select>
                      <button class="myButton" ng-click="updateCommodity(commodity._id)">
update</button>
                 </span>
            </td>
            <td><button class="myButton" ng-click='deleteCommodity(commodity._id)'>
delete</button></td>
</tr>
```

In this way, getCommodities() is executed when the application is initialized, and the information of commodities received from the server is displayed underneath "Commodity List" on the page. Note that in the table cell displaying the status of the commodity, we use two <span> elements and ng-hide/ng-show directives to decide which elements to display. We have also added a few event handlers for "ng-click" events on the <button> elements, which we will implement in Task 5 and Task 6.

Browse the web page at http://localhost:3000/index.html. You should see a page as follows:



Category	Name	Status	Delete?
Computer	lenovo	in stock update	delete

Fig. 2

Task 4: We next add code for adding a new commodity into the database. In **externalJS.js**, add the following code in the constructor of the "commodityListController" controller.

```
$scope.new_commodity = {categoty:"", name:"", status:""};

$scope.addNewCommodity = function(){
  if($scope.new_commodity.name==''||$scope.new_commodity.category==''||
$scope.new_commodity.status==''){
        alert("please fill in all fields");
        return;
      }

      $http.post(?, ?).then(function(response){
        if(response.data.msg===''){
          $scope.new_commodity = {category:"", name:"", status:""};
          $scope.getCommodities();
        }
        else{
          alert("Error adding commodity:"+response.data.msg);
        }
      }, function(response){
          alert("Error adding commodity:"+response.statusText);
      });
};
```

Replace "?" with the correct code to finish the client-side code for sending an AJAX HTTP POST request for http://localhost:3000/users/ addcommodity. Recall that the end point is handled by the following middleware in ./routes/users.js.

```
/*
* POST to add commodity
*/
router.post('/addcommodity', function (req, res) {
  var db = req.db;
  var collection = db.get('commodities');

  //insert new commodity document
  collection.insert(req.body, function (err, result) {
        res.send(
          (err === null) ? { msg: '' } : { msg: err }
        );
  });
});
```

Add the following three directives in the text <input> element and <select> elements in **index.html** for Name, Category and Status, respectively. The ng-model directives bind values of the input and select elements with the variables in the new_commodity object.

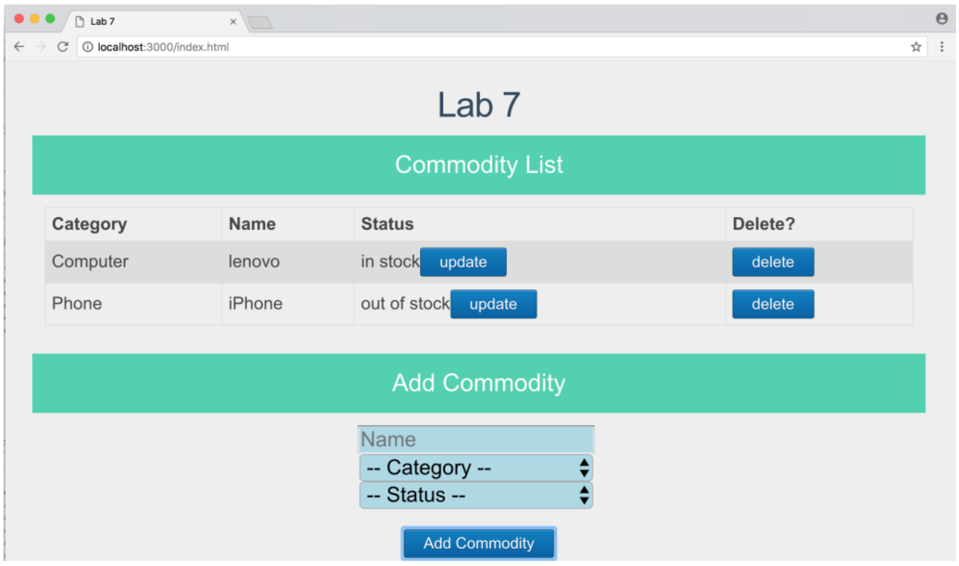
```
ng-model="new_commodity.name"

ng-model="new_commodity.category"

ng-model="new_commodity.status"
```

Add the following directive in the "Add Commodity" <button> element, to register addNewCommodity () as the event handler for "ng-click" on the button.

```
ng-click="addNewCommodity()"
```

The above codes in **externalJS.js** and **index.html** achieve the following functionalities: after the new commodity's information is entered and the "Add Commodity" button is clicked, an AJAX HTTP POST request is sent to the server, carrying the new commodity's information. After receiving a success response from the server, the commodity list display is refreshed and the commodity information in the input textboxes is cleared.

Restart your Express app and browse http://localhost:3000/index.html again. Add information of a new commodity. After clicking the "Add Commodity" button, you should see a page as in the figure below.



Lab 7

Commodity List

Category	Name	Status	Delete?
Computer	lenovo	in stock update	delete
Phone	iPhone	out of stock update	delete

Add Commodity

Fig. 3

Task 5: Now we implement the code for deleting a commodity from the database, when a respective "delete" button in the Commodity List is clicked. Recall that delete is handled by the following middleware in ./routes/users.js:

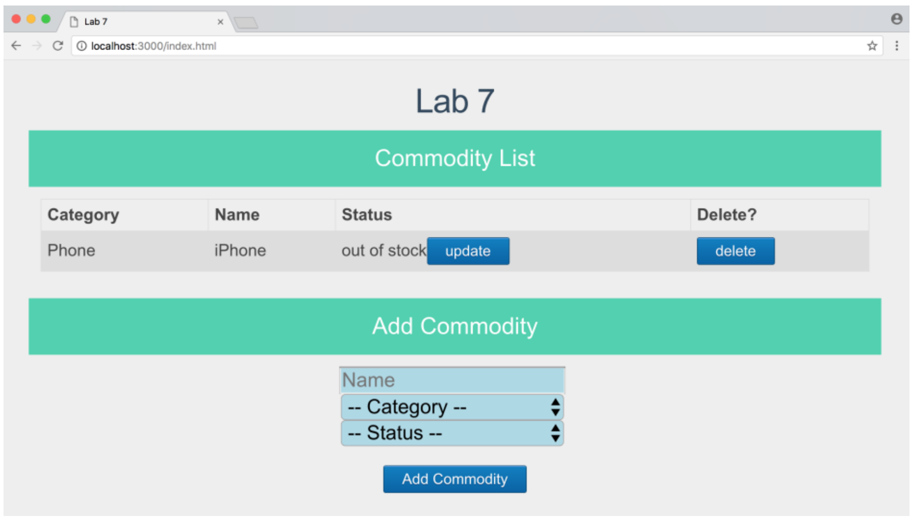
```
/*
* DELETE to delete a commodity.
*/
router.delete('/deletecommodity/:id', function (req, res) {
  var db = req.db;
  var collection = db.get('commodities');
  var commodityToDelete = req.params.id;

  collection.remove({'_id': commodityToDelete}, function (err, result) {
    res.send(
      (err === null) ? { msg: '' } : { msg: err }
    );
  });
});
```

In **externalJS.js**, add the following code:

```
    $scope.deleteCommodity = function(id){
        var url = ?;

        $http.?.then(function(response){
            ?
        }, function(response){
            alert("Error deleting commodity:"+response.statusText);
            });
    };
```

Replace "?" with correct code to complete the client-side code for sending an AJAX HTTP DELETE request and handling the response. Your implementation should achieve the following: upon successful deletion, you should refresh the "Commodity List" displayed on the web page; otherwise, display the error message.

Restart your Express app, browse http://localhost:3000/index.html again, and test the delete function as follows:



Category	Name	Status	Delete?
Phone	iPhone	out of stock	

Fig. 4 After deleting the “lenovo” commodity

Task 6: Now we implement the code for updating the status of an existing commodity in the database. Recall that status update is handled by the following middleware in ./routes/users.js:

```
/*
* PUT to update a commodity (status)
*/
router.put('/updatecommodity/:id', function (req, res) {
  var db = req.db;
  var collection = db.get('commodities');
  var commodityToUpdate = req.params.id;

  var filter = { "_id": commodityToUpdate};
  collection.update(filter, { $set: {"status": req.body.status}}, function (err, result) {
              res.send(
                (err === null) ? { msg: '' } : { msg: err }
              );
  })
});
```

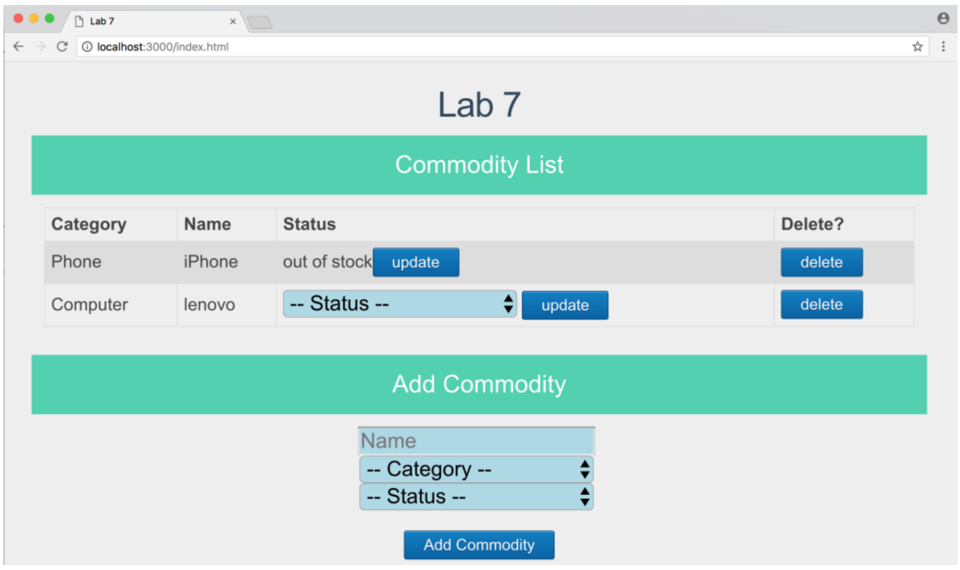
In **externalJS.js**, add the following code:

```
     $scope.showStatusOptions = function(id){
         $scope.updateStatusOrNot[id]=true;
      }

      $scope.updateCommodity = function(id){
         if ($scope.new_status[id] === ""){
             alert('Please select status');
             return false;
         }
         else{
             var changeStatus = {
                 ?
             }
```

```
            $http.put(?, changeStatus).then(function(response){
               ?
            },function(response){
               alert("Error updating commodity:"+response.data.msg);
            });
        }
    }
```

Replace "?" with correct code to finish the client-side code for sending an AJAX HTTP PUT request and handling the response. Especially, upon successful update, you should refresh the "Commodity List" display on the web page; otherwise, prompt the error message carried in the response using alert(). (Note: You do not need to handle the case that the newly selected status is in fact the same as the old status.)

Restart your Express app, browse http://localhost:3000/index.html again, and test the update function as follows:



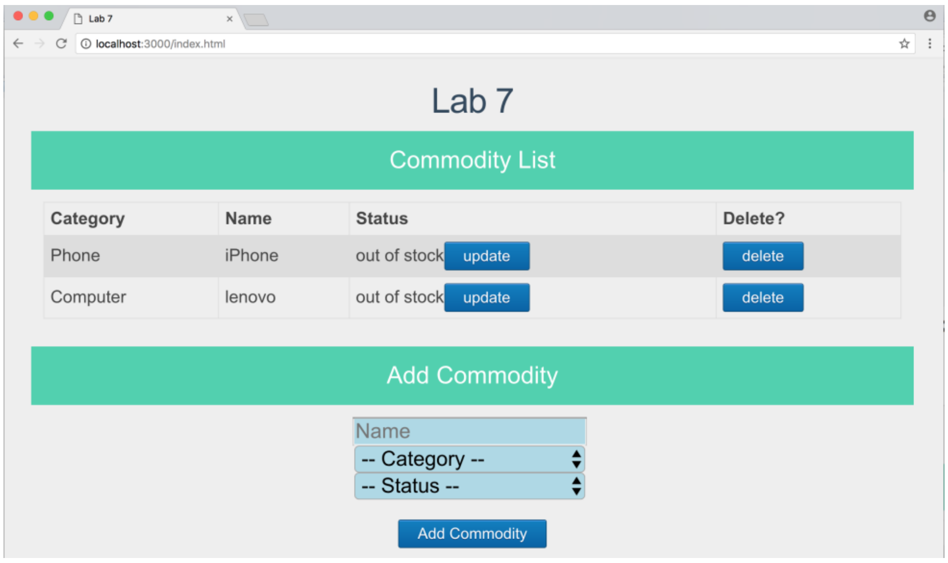
Lab 7

Commodity List

Category	Name	Status	Delete?
Phone	iPhone	out of stock update	delete
Computer	lenovo	-- Status -- update	delete

Add Commodity

Fig. 5 After clicking "update" button in the row of "Computer lenovo in stock"



Lab 7

Commodity List

Category	Name	Status	Delete?
Phone	iPhone	out of stock update	delete
Computer	lenovo	out of stock update	delete

Add Commodity

Fig. 6 After selecting "out of stock" for "Computer lenovo" and then clicking "update" button

Submission:

Create a .zip file named **lab7.zip** which should contain **app.js**, **index.html** and **externalJS.js**. Please upload lab7.zip to i.cs.hku.hk web server under “**lab7**” before **23:59 Sunday April 15, 2018**. The URL to access your submission should be **http://i.cs.hku.hk/~[your_CSID]/lab7/lab7.zip**. We will check your files for lab 7 marking.