# COMP3322A Modern Technologies on World Wide Web

## Lab4: NodeJS, AJAX and MongoDB basics

### Overview

In this lab, we will develop a simple web-based stock status system, with which we can maintain a stock list. As shown in Fig. 1, each stock entry includes the status (RISE or FALL), stock name, stock category, stock code and date. We will practice loading the list from the database and toggling between the RISE status and the FALL status. We will also implement a few buttons to decide the (selected) entries to be displayed. For example, when "MSFT" is typed into the textbox above "Filter by Stockcode" button, the page shows up as in Fig. 2; when "Bank" is typed into the textbox above "Filter by Category" button, the page shows up as in Fig. 3. Note that a "Show All" button is displayed in Fig. 2 and Fig. 3, and when it is clicked, the page view goes back to Fig. 1.

These functionalities will be implemented through server-side implementation (ExpressJS) and the client-side implementation (HTML, JavaScript, AJAX). We adopt MongoDB as the back-end database server to store the stock information.

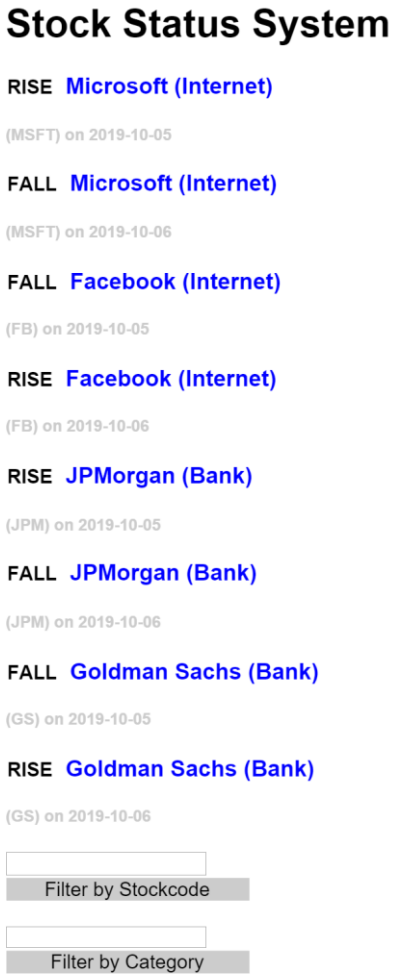


**Fig. 1**


**Fig. 2**

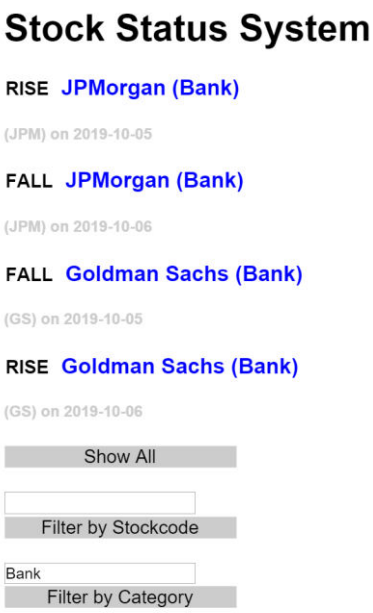


**Fig. 3**

## Set Up Runtime Environment and MongoDB

Follow the instructions (Steps 1 to 3) in **setup_nodejs_runtime_and_examples_1.docx** to set up Node.js runtime environment and create an Express project named "**lab4**" (i.e., use "express lab4" in Step 3).

Follow the following steps (similar to what you did when trying out Example 3 in **AJAX_JSON_MongoDB_setup_and_examples.docx)** to prepare the MongoDB server and database.

[Step1]: In the "lab4" project directory, create a new directory "data". This directory will be used to store database files.

```
cd lab4
mkdir data
```

[Step2]: Go to https:/ / www.mongodb.com/ download-center/ community and download the latest version of MongoDB (choose the latest "Community Server" release to download). Install MongoDB to a directory at your choice.

[Step3]: **Launch the 2nd terminal** (besides the one you use for running NPM commands), and switch to the directory where MongoDB is installed. Start MongoDB server using the "data" directory of "lab4" project as the database location, as follows: (replace "YourPath" by the actual path on your computer that leads to "lab4" directory)

```
./bin/mongod --dbpath YourPath/lab4/data
```

After starting the database server successfully, you should see some prompt in the terminal like "…2019-10-08T21:49:47.404+0800 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 when your Express app is running** in order to allow connections to the database from your Express app.

[Step4]: **Launch the 3rd terminal**, switch to the directory where mongodb is installed, and execute the following commands:

```
./bin/mongo
use lab4
db.stockList.insert({'stockname':'Microsoft', stockcode:'MSFT', 'category':'Internet',
'date':'2019-10-05', status:'RISE'})
db.stockList.insert({'stockname':'Microsoft', stockcode:'MSFT', 'category':'Internet',
'date':'2019-10-06', status:'FALL'})
db.stockList.insert({'stockname':'Facebook', stockcode:'FB', 'category':'Internet',
'date':'2019-10-05', status:'FALL'})
db.stockList.insert({'stockname':'Facebook', stockcode:'FB', 'category':'Internet',
'date':'2019-10-06', status:'RISE'})
db.stockList.insert({'stockname':'JPMorgan', stockcode:'JPM', 'category':'Bank',
'date':'2019-10-05', status:'RISE'})
db.stockList.insert({'stockname':'JPMorgan', stockcode:'JPM', 'category':'Bank',
'date':'2019-10-06', status:'FALL'})
db.stockList.insert({'stockname':'Goldman Sachs', stockcode:'GS', 'category':'Bank',
'date':'2019-10-05', status:'FALL'})
db.stockList.insert({'stockname':'Goldman Sachs', stockcode:'GS', 'category':'Bank',
'date':'2019-10-06', status:'RISE'})
```

The “use lab4” command creates a database named “lab4”. The next command followed by “use lab4” inserts a new record into the “stockList” collection in the database.

After you run the insert command, you should see “WriteResult({ "nInserted" : 1 })” on the terminal. You can insert more records into the database collection to facilitate testing of your program, if needed.

[Step 5]: Switch back to the 1st terminal which you use for running NPM commands. Install the monk module for your express app project with the following command:

```
npm install monk
```

## Lab Exercise 1 - Load entries from the database

Download **lab4_materials.zip** from Moodle. Unzip it and you will find two files needed on the client side for this app.

Copy **index.html** to the “public” folder of the project directory and copy **style.css** to the “public/ stylesheets” folder. Open index.html with a text editor. You will see it contains the following HTML content:

```
<!DOCTYPE html>
<html>
```

```
<head>
  <title>Stock Status System</title>
     <link rel="stylesheet" href="/stylesheets/style.css">
</head>

<body>
        <h1>Stock Status System</h1>
        <div id="List">
                <div id="entries">
                </div>

                <div id="button_all" class="buttons">
                        <p> Show All</p>
                </div>

                <input id="stockcode" type="text">
                <div class="buttons">
                        <p>Filter by Stockcode</p>
                </div>

                <input id="category" type="text">
                <div class="buttons">
                        <p>Filter by Category</p>
                </div>
        </div>
  <script>
    //to be implemented

  </script>
 </body>
</html>
```

### Task 1. Implement client-side code for loading all entries

When the page is loaded or the "Show All" button (on the page views as in Fig. 2 or Fig. 3) is clicked, load and display all the stock entries from the stockList collection in the database, using AJAX.

[Step1]: In **index.html**, identify the events that initiate the AJAX communication:

1. <body> is loaded. **Event**: onload.
2. <p> Show All</ p></ div> is clicked. **Event**: onclick.

[Step2]: Define the event handler (JavaScript function): showAll() for both events.

In the event handler, create an XMLHttpRequest object.

```
<script>
  function showAll(){
    var xmlhttp = new XMLHttpRequest();

   //More code here for showing all entries
  }
```

```
</script>
```

[Step3]: Define the response actions when the server's response is received.

```
xmlhttp.onreadystatechange = function(){
  if (xmlhttp.readyState == 4  && xmlhttp.status ==200){
    var stocks = document.getElementById("entries");
    stocks.innerHTML = xmlhttp.responseText;
  }
}
```

[Step4]: Define the request that is sent "behind the scenes":

```
xmlhttp.open("GET","GetEntries?show=all", true);
```

[Step5]: Send the request.

```
xmlhttp.send();
```

[Step6]: Hide the "Show All" button from the full list display view (as in Fig. 1), as follows:

```
document.getElementById("button_all").style.display = "none";
```

## Task 2. Implement client-side code for loading selected entries based on "Stockcode" or "Category"

When the "Filter by Stockcode" division is clicked, load and display all the stock entries whose stockcode matches the entered text in the textbox above it, using AJAX.

[Step1]: In **index.html**, identify the event that initiates the AJAX communication.

[Step2]: Define the event handler (JavaScript function): **filterS()** for the event. In the event handler, create an XMLHttpRequest object and define the response actions when the server's response is received, following steps 2 and 3 in Task 1.

Define the request that is sent "behind the scenes" as follows:

```
xmlhttp.open("GET","GetEntries?show=stockcode&value="+stockCode, true);
```

Here stockCode should be the value user enters in the textbox above the "Filter by Stockcode" division.

Send the request following step 5 in Task 1.

[Step3]: Display the "Show All" button in the page view (as in Fig. 2), learning from what we did in step 6 of Task 1.

[Step4]: Similar to the above steps, implement the client-side code to achieve the functionality that when "Filter by Category" division is clicked, load and display all the stock entries whose category matches the entered text in the textbox above the division, using AJAX. The request to be sent "behind the scenes" is as follows:

```
xmlhttp.open("GET","GetEntries?show=category&value="+category, true);
```

Here category should be the value user enters in the textbox above the "Filter by Category" division.

## Task 3. Implement server-side code for loading entries from the database

We next implement our Express app server-side code to connect to the database and retrieve entries from the stockList collection.

[Step1]: Open **app.js** and change its content to the following:

```
var express  = require('express');
var app = express();

var monk = require('monk');
var db = monk('localhost:27017/lab4');

// allow retrieval of static files and make db accessible to router
app.use(express.static('public'), function(req,res,next){
    req.db = db;
    next();
})


var server = app.listen(8081, function () {
  var host = server.address().address
  var port = server.address().port
  console.log("Example app listening at http://%s:%s", host, port)
})
```

[Step2]: Add the middleware to handle HTTP "**GetEntries**" requests (i.e., retrieve all documents, or documents whose stockcode or category matches the received value), as follows:

```
app.get('/GetEntries', function(req, res){
        var db = req.db;
        var collection = db.get('stockList');
        var show = req.query.show;
        var value = req.query.value;
        if (show == "all"){
```

```
        collection.find({}, {}, function(err, docs){
            if (err === null){
                res.send(ResHTML(docs));
            } else res.send(err);
        });
    }else if (show == "category"){
        // TODO: retrieve documents whose category matches the received value, construct the HTML text accordingly using ResHTML(), and send the response
    } else if (show == "stockcode"){
        // TODO: retrieve documents whose stockcode matches the received value, construct the HTML text accordingly using ResHTML(), and send the response
    }
});
```

The function ResHTML() for constructing the HTML text based on retrieved documents is given as follows:

```
function ResHTML(docs) {
        var response_string = "";
        for (var i = 0; i < docs.length; i++) {
                var stock = docs[i];
                response_string += "<div id="+stock['_id']+">";
                response_string += "<span onclick=\"changeState(this)\">" + stock['status'] + "</span><h3>" + stock['stockname'];
                response_string += " (" + stock['category'] + ")</h3><h5>(" + stock['stockcode'] + ") on " + stock['date'] + "</h5>";
                response_string += "</div>";
        }
        return response_string;
}
```

Implement the code in the above middleware, for retrieving specific stock records in the stockList collection according to client's request. Upon successful retrieving, the server should send a response message back to the client with the "HTML" text containing the stock records.

## Lab Exercise 2 – Toggle RISE/FALL

Tapping the word "RISE" or "FALL" in each entry on the web page will change the status from one to the other, and initiate AJAX communication to the server for changing the attribute "status" of this entry in the database, without reloading the entire web page.

### Task 1 Client-side Implementation

An event handler changeState(this) has been registered with the onclick event on the  element of each loaded stock entry (see the code in function ResHTML()). Implement the event handler function changeState(elem) (as partially given below) in **index.html**, by completing the AJAX code which sends an HTTP POST request to the server side for "/ updateState", carrying the newvalue and itemID in the body of the request, and displaying the received new status in the stock entry after successfully receiving server response.

```
function changeState(elem) {
        var oldValue = elem.innerHTML;
        var newvalue;
        var itemID = elem.parentNode.getAttribute('id');

        if (oldValue == 'RISE') {
                newvalue = 'FALL';
        } else {
                newvalue = 'RISE';
        }
        var xmlhttp;
        // Add AJAX code here
}
```

### Task 2 Server-side Implementation

Open app.js and add a middleware to implement the server-side logic of changing the "status" attribute of the corresponding stock document in the database. The middleware should handle POST requests for "updateState". The following code is given for your reference.

```
        var id = req.body.id;
        var newValue = req.body.newValue;

        collection.update({"_id":id}, {$set:{'status':newValue}}, function(err, docs){
                if (err === null) {
                        res.send(newValue);
                } else res.send(err);
        })
```

Read https:/ / automattic.github.io/ monk/ docs/ collection/ update.html to learn more about collection.update with monk.

## Submission:

You should submit the following files only:
(1) app.js
(2) index.html

Please compress the above two files in a .zip file and submit it on Moodle before 23:59 Wednesday Oct. 23, 2019:

(1) Login Moodle.
(2) Find “Labs” in the left column and click “Lab 4”.
(3) Click “Add submission”, browse your .zip file and save it. Done.
(4) You will receive an automatic confirmation email, if the submission was successful.
(5) You can “Edit submission” to your already submitted file, but ONLY before the deadline.