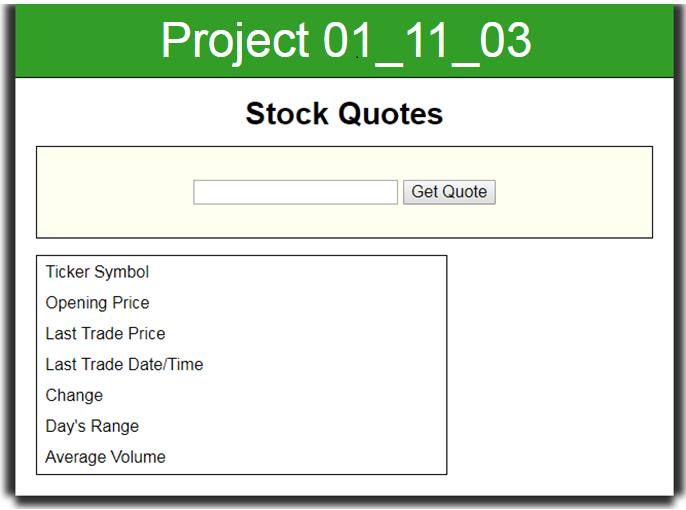
**Project 01\_11\_03**

As you have seen the Stock quote program that we built is not working correctly. In this project, you will modify the code from the previous project to use a completely different Web service.

1. Copy all of the completed files from ***Project 01\_11\_02*** into a ***Project 01\_11\_03*** folder; this will contain all of the files you will zip up and submit. Open the **stocks.html** file and change the **<h1>** content as follows:  
    *<header>  
    <h1>* ***Project 01\_11\_03*** *</h1>  
    </header>*
2. Let’s debug the project. Open op the **scrtipt.js** file and change the documentation to **Project 01\_11\_03**. Go to the **displayData()** function and place the **console.log()** debug back into the code as follows:  
   *function displayData() {*  
    *if (httpRequest.readyState === 4 && httpRequest.status === 200) {  
    var stockResults = httpRequest.responseText;* ***console.log(stockResults);*** *}  
   }*  
   A server test with Developer Tools in the Console tab should let us test. Make sure there are no syntax errors. The ***console.log()*** data should show us that we got garbage, a mess of HTML.
3. So are we actually getting data in from the server? Let’s put the following debug code into **displayData()** to monitor or **readyState** and **status** to see how the AJAX executes:  
   function displayData() {  
    **console.log("httpRequest.readyState: " + httpRequest.readyState);  
    console.log("httpRequest.status: " + httpRequest.status);**  
   In a server test, it appears that our readyState climbs directly through the proper values and gives us successful status. But we get a garbage response.
4. Has something changed with Yahoo Finance? That would certainly not be unusual. Google “**yahoo finance quotes api**” to see what we can find. Sure enough, in the results we find a “**Bye Yahoo, and thanks for all the fish – The Financial Hacker**” category. Click on that and let’s see what we see. Wow “**aargh!**” and welcome to the life of a coder. The API has disappeared.
5. Let’s see if we can find another API to use for stock quotes. Google “**yahoo finance quotes api replace**” to see what we can find. In the results we find a “**Replacing Yahoo Finance with the IEX API and Alpha Vantage**” category. Click on that and we get a couple of choices. Click on the “**Alpha Vantage API**” link and let’s look it over. Ok, free, JSON format, needs an API key. Let’s look at the documentation. Looks good, but lots of API categories, and the **endpoints** look a little complex, but a definite possibility.
6. Let’s look at the other one. Click on the “**IEX**” link and let’s look it over. Simple menu on the left, so we click on **endpoints**. Says that all endpoints are prefixed with:  
   **https://api.iextrading.com/1.0**  
   Also says supports JSON and there is no mention of an API key. So far, so good.
7. Let’s click on the Stocks option and see what is available. First thing it tells us is to “use the **/ref-data/symbols** endpoint to find the symbols that we support” which sounds pretty simple. Let’s go into **StockCheck.php** and copy lines 4 & 5 to make some changes and just test this API as follows:  
   **//header("Content-Type: text/csv");  
   //$Quote = "http://quote.yahoo.com/d/quotes.csv?s=$TickerSymbol&f=  
    sl1d1t1c1p2ohgv";  
   header("Content-Type: application/json");  
   $Quote = "https://api.iextrading.com/1.0/ref-data/symbols";**  
   Give this a server test and we have some clean JSON with stock symbols.
8. Let’s do some further experimentation in **Stocks** and see what kind of data we can get that meets the needs of our Website. Scroll down past **Batch** **Requests** and we see some options under **Book**. It further tells us that we can get responses from ***quote*** and give it a stock symbol. Let’s check it out by clicking on it. The explanation and JSON look like just what we need. Change the StockCheck.php **endpoint** as follows:  
   **$Quote =   
    "https://api.iextrading.com/1.0/stock/$TickerSymbol/quote";**  
   Let’s give that a server test. It appears that we get a status **404**.If we look at the request, we see a weird looking symbol. Seems to indicate our symbol is not found.
9. The symbol **^IXIC** is bogus; it is for Yahoo, indicating we don’t want an individual stock, we want the whole market. Not supported by IEX. Let’s make a change to **Microsoft** in our global variables in script.js:  
   /\* global variables \*/  
   var httpRequest = false;  
   **var entry = "MSFT";**  
   This appears to work based on or Console data. So we go to the boss, and tell him about Yahoo. Tell him not to worry, we now have much more data. We show him and say we are going to improve the Website, and of course charge some more.
10. For starters, let’s get our starting stock into the search box. Go to the ***getQuote()*** function, and let’s add an else statement to originally populate the control:  
    function getQuote() {  
     if (document.getElementsByTagName("input")[0].value) {  
     entry = document.getElementsByTagName("input")[0].value;  
     }  
     **else {  
     document.getElementsByTagName("input")[0].value = entry;  
     }**  
    Go to **displayData()** and let’s remove our **readyState** and **status** debug and give this a test. The search box should populate.
11. We can see that our response is a string representation of JSON. Let’s convert that to JavaScript JSON so we can use it. We change the **array.split()** function to a **JSON.parse()**. We will also remove the ***for*** loop that formats an array, we don’t need it anymore. Lastly, let’s **console.log()** our converted string:  
     var stockResults = httpRequest.responseText;  
     console.log(stockResults);  
     **var stockItems = JSON.parse(stockResults);  
     console.log(stockItems.symbol);** document.getElementById("ticker").innerHTML = stockItems[0];  
    Now we have clean JavaScript JSON to work with.
12. Let’s populate our first data field for a test as follows:  
     console.log(stockItems.symbol);  
     **document.getElementById("ticker").innerHTML = stockItems.symbol;**  
    Give it a test and it is working well.
13. Let’s get the rest of the data as follows:  
     document.getElementById("ticker").innerHTML = stockItems.symbol;  
     **document.getElementById("openingPrice").innerHTML =   
     stockItems.open;  
     document.getElementById("lastTrade").innerHTML =   
     stockItems.latestPrice;  
     var date = new Date(stockItems.latestUpdate);  
     document.getElementById("lastTradeDT").innerHTML =   
     date.toDateString() + "<br>" + date.toLocaleTimeString();  
     document.getElementById("change").innerHTML =   
     (stockItems.latestPrice - stockItems.open).toFixed(2);  
     document.getElementById("range").innerHTML = "Low " +   
     (stockItems.low \* 1).toFixed(2) + "<br>High " +   
     (stockItems.high \* 1).toFixed(2);  
     document.getElementById("volume").innerHTML =   
     (stockItems.latestVolume \* 1).toLocaleString();**Give this a test. When it works, remove the debug.
14. Lastly, let’s cite IEX as required in their Terms of Use. Add the following to **stocks.html**:  
     <input type="submit" value="Get Quote" />  
     **<br><br>  
     <p>  
     Data provided for free by   
     <a href="https://iextrading.com/developer/">IEX</a>  
     </p>  
     <p>  
     View   
     <a href="https://iextrading.com/api-exhibit-a/">  
     IEX’s Terms of Use</a>  
     </p>**Give this a final test, and it appears that it is a wrap.