**Exercise 01\_11\_01 – Step 1**

In this Exercise, we will learn how to open files from a web server. we will also learn how to retrieve dynamic data from the server to update the client application. We will make use of AJAX to accomplish this.

 

1. Copy all of the completed files from ***Exercise 01\_10\_01.zip*** into an ***Exercise\_01\_06\_01*** folder; this will contain all of the files you will zip up and submit.
2. Open ***solar.html***; fill in the appropriate documentation in the top comments. Scroll through the document to get familiarized with the content. Notice that the file contains a mostly empty ***<table>*** structure. We will use JavaScript, in combination with AJAX, to write in most of the content dynamically.
3. Open the JavaScript document named ***script.js***. Add all of the appropriate documentation to the top of the file, with your name and the date. The beginnings of the script have been set up.
   1. There are a couple of global variables, a function named ***getWeather()***, and the script creates some event listeners.
   2. The event listeners are set up to call ***getWeather()*** on the page ***load*** event and on some ***click*** events.
   3. We will employ ***AJAX*** to retrieve current data from the server so that we can use it to populate the web page.
4. In order to build and employ a client-server environment, we are going to need to copy all of our files to a server domain. Copy your entire project folder to the server, into its root domain. This will create a sub-domain named Exercise 01\_10\_01. Depending on your computer environment, editing your files and running them on the server may be different. On a Windows XAMPP stack, we can edit directly in the server structure. On MAC, we will have to edit on the local computer then move the new files over to the server to test them.
5. Once the files are moved, with the server running, navigate to the new domain and run ***solar.html*** to take a look around. Pretty, but not much going on yet.

**Exercise 01\_11\_01 – Step 2**

 

1. Go to the server domain for the project and run ***solar.html***. Open the ***Developer Tools*** and go to the ***Network*** tab. This tab shows details about each HTTP request and response related to the current page. Refresh the page if necessary, the request only registers when the tools are open, so it might start off empty. In Chrome, hit ***F5*** for a list of the ***messages***.

**Exercise 01\_11\_01 – Step 3**

 

1. Go to the server domain for the project and run ***solar.html***. Open the ***Developer Tools*** and go to the ***Network*** tab. This tab shows details about each HTTP request and response related to the current page. Refresh the page if necessary and hit ***F5*** for a list of the ***messages***. Make sure the sort order is correct, from first to last
2. Find the ***solar.html*** message and click it. the summary of the messages is replaced with the detail of the clicked message.
3. Examine the detail of the message. You can see ***General*** details of the message. This is followed by both ***Response*** and ***Request*** ***Headers***. Examine the ***Request*** Headers information in detail.

**Exercise 01\_11\_01 – Step 4**

 

1. Go to the server domain for the project and run ***solar.html***. Open the ***Developer Tools*** and go to the ***Network*** tab. This tab shows details about each HTTP request and response related to the current page. Refresh the page if necessary and hit ***F5*** for a list of the ***messages***. Make sure the sort order is correct, from first to last.
2. Find the ***solar.html*** message and click it. the summary of the messages is replaced with the detail of the clicked message.
3. Examine the detail of the message. You can see ***General*** details of the message. This is followed by both ***Response*** and ***Request*** ***Headers***. Examine the ***Response*** Headers information in detail.

**Exercise 01\_11\_01 – Step 5**

 

1. We are going to used a server based PHP file as a proxy for our AJAX response. On the client side, open ***solar.php*** in your IDE. Let’s examine the contents.
   1. First notice the opening and closing PHP tags: ***<?php*** and ***?>***. The indicate that what is enclosed is a PHP script. PHP scripts are often embedded in HTML files with .php extensions.
   2. Line 2 creates a PHP variable ***$WeatherSource***. PHP variables begin with the ***$*** sign.
   3. The variable is a string variable, the PHP ***concatenation*** operator is the period (***.***).
   4. The first part of the string is a ***URL*** to access a ***web*** ***service*** that we will use for the weather forecast. Notice the ***/apikey/*** part. We will substitute a key in there that will allow access to the service.
   5. ***$\_GET*** is a global PHP variable. It holds the contents of whatever data was sent by the browser ***HTTP GET request*** for this resource. The data is sent in **name/value pairs**, so the $\_GET parameter holds the name of the pair that is of interest.
   6. Next are two PHP ***header()*** functions, here being used to set two ***response*** ***header*** ***pairs***.
   7. The PHP ***readfile()*** function is being used to return the forecast data as the ***body*** of the ***HTTP response***. Let’s go to the forecast site at ***developer.forecast.io***.
2. Now we have to obtain a value for the ***API*** ***key*** to get access to the web service. We will all have our own. Click the ***Sign Up*** button, enter an email address and password, and get an account. Make sure that you remember this sign in. when you are done, you will receive your personal key.
3. Make a copy of the key and go back to ***solar.php***. Paste the key over the text ***apikey*** in the URL string. Save the file and move a copy over to your web domain.

**Exercise 01\_11\_01 – Step 6**

 

1. Open the ***script.js*** file in the IDE. We will first add a ***httpRequest*** variable to the global variables. It will be used to track whether or not we have an existing request open. This will facilitate reusing the XHR object to optimize page loading:  
   ***var httpRequest = false;***
2. Below the globals, let’s create a function that will handle XHR instantiation. Include an ***alert()*** for testing:  
   ***function getRequestObject() {  
    alert("getRequestObject()");   
   }***
3. Let’s give it a test by calling the new function from the ***getWeather()*** function. Just before the closing, add the following code:  
    ***if (!httpRequest) {  
    httpRequest = getRequestObject();  
    }***  
   Let’s move that code to the server and give it a browser test. We should get an alert indicating our function is being called.
4. Now let’s build this into a ***try*** / ***catch*** structure to give us some protection:  
    ***try {  
    httpRequest = new XMLHttpRequest();   
    }  
    catch (requestError) {  
    document.querySelector("p.error").innerHTML = "Forecast not   
    supported by your browser.";  
    document.querySelector("p.error").style.display = "block";  
    return false;  
    }  
    return httpRequest;***  
   Let’s give this a browser test for both syntax and to see if any errors are thrown.

**Exercise 01\_11\_01 – Step 7**

 

1. Open the ***script.js*** file in the IDE. We now need to target the server resource and send a request. We will add this code to the bottom of the ***getWeather()*** function. We will first ***abort()*** any outstanding requests, we will then construct our URL string in the ***open()*** method, and we will finally ***send()*** our request with no body:   
    ***httpRequest.abort();  
    httpRequest.open("get", "solar.php?" + "lat=" + latitude +   
    "&lng=" + longitude, true );  
    httpRequest.send(null);***Save and copy the modified file to your server domain.
2. In the browser, turn on the ***Developer*** ***Tools*** and go to the ***Network*** tab. Load or refresh ***solar.html***. One of the messages should be marked ***solar.php?...*** Look at the ***Headers*** tab and under ***General*** there should be a well-formed URL with the correct ***name-value pairs***. Look at the ***Response Headers*** and you should see the proper ***Cache-Control*** and ***Connection*** pairs that were set in the solar.php file.
3. Switch to the ***Console*** tab. Type in ***httpRequest.responseText*** and press ***Enter***. Chrome has good auto-complete for this. You should see a long string of data. This is your ***response*** from the ***web service***. We will have to learn how to process that next.

**Exercise 01\_11\_01 – Step 8**

 

1. Open the ***script.js*** file in the IDE. We now need to Process the response data. To do this we will build a new function directly below the ***getWeather()*** function:   
   ***function fillWeather() {  
      
   }***
2. Now let’s build out the function, looking for a ***readyState*** of ***4*** indicating data is ***loaded*** and a ***status*** of ***200***, indicating request was ***successful***. Both must be true to process the data. We will then convert the ***JSON*** data to a JavaScript object:  
    ***if (httpRequest.readyState === 4 &&  
    httpRequest.status === 200) {  
    weatherReport = JSON.parse(httpRequest.responseText);  
    }***
3. Next we have to set up the ***XMLHttpRequest*** ***event*** property to use the new function as its event handler. We will do this as the last command in the getWeather function:  
    ***httpRequest.onreadystatechange = fillWeather;***Let’s move this over to the web server domain.
4. In the browser, turn on the ***Developer*** ***Tools*** and go to the ***Console*** tab. Load or refresh ***solar.html***. Type in ***weatherReport*** and press ***Enter***. The object sitting in that variable is displayed. Expand the ***triangle*** next to it. This will display the contents of the object, name-value pairs. Expand each of the properties and you can clearly see the weather data.

**Exercise 01\_11\_01 – Step 9**

 

1. Open the ***script.js*** file in the IDE. We now need to get the processed data onto the page. To do this we will build out some code that uses DOM access to fill in the data to the HTML elements. In the ***fillWeather()*** function, at the bottom of the if statement, let’s first build some variables to support this task:   
    ***if (httpRequest.readyState === 4   
    && httpRequest.status === 200) {  
    weatherReport = JSON.parse(httpRequest.responseText);  
    var days = ["Sunday","Monday","Tuesday",  
    "Wednesday","Thursday","Friday","Saturday"];  
    var dateValue = new   
    Date(weatherReport.daily.data[0].time);  
    var dayOfWeek = dateValue.getDay();  
    var rows = document.querySelectorAll  
    ("section.week table tbody tr");  
    document.querySelector("section.week table caption").  
    innerHTML = selectedCity;  
    }***Move this over to the domain, run ***solar.html***, and check the Console for syntax or other errors.
2. Ok now for some pretty detailed code to populate that document. Let’s take it in small enough bites so that we do not get into trouble. First, let’s see if we can get the table to display and test our above code:  
    ***document.querySelector("section.week table   
    caption").style.display = "block";  
    document.querySelector("section.week   
    table").style.display = "inline-block";***  
   Let’s test that on the server. We should get our selected city showing and 7 days of sun symbols that come from our HTML file.
3. Now we can build ourselves a ***for*** loop to continue populating those ***<table>*** rows. We will place that directly above the two previous statements:  
    ***for (var i = 0; i < rows.length; i++) {  
    var firstCell =   
    rows[i].getElementsByTagName("td")[0];  
    var secondCell =   
    rows[i].getElementsByTagName("td")[1];  
    var thirdCell =   
    rows[i].getElementsByTagName("td")[2];  
    firstCell.innerHTML = days[dayOfWeek];  
    }***  
   A server test of this code should give us at least one of the days of the week displayed on each row. We will fix that next.
4. Let’s fix up the day for the rows by calculating on each iteration directly below the previous code in the ***for*** loop:  
    ***if (dayOfWeek + 1 === 7) {  
    dayOfWeek = 0;  
    }  
    else {  
    dayOfWeek++;  
    }***  
   A server test of this should give us the proper display of a week.
5. Now let’s complete our <table> cells for each row  
    ***var sun = Math.round((1 -   
    weatherReport.daily.data[i].cloudCover) \*100, 0)  
      
    if (sun > 90) {  
    secondCell.style.color = "rgb(255,171,0)";  
    }  
    else if (sun > 80 && sun <= 90) {  
    secondCell.style.color = "rgb(255,179,25)";  
    }  
    else if (sun > 70 && sun <= 80) {  
    secondCell.style.color = "rgb(255,188,51)";  
    }  
    else if (sun > 60 && sun <= 70) {  
    secondCell.style.color = "rgb(255,196,77)";  
    }  
    else if (sun > 50 && sun <= 60) {  
    secondCell.style.color = "rgb(255,205,102)";  
    }  
    else if (sun > 40 && sun <= 50) {  
    secondCell.style.color = "rgb(255,213,128)";  
    }  
    else if (sun > 30 && sun <= 40) {  
    secondCell.style.color = "rgb(255,221,153)";  
    }  
    else if (sun > 20 && sun <= 30) {  
    secondCell.style.color = "rgb(255,230,179)";  
    }  
    else if (sun > 10 && sun <= 20) {  
    secondCell.style.color = "rgb(255,238,204)";  
    }  
    else if (sun <= 10) {  
    secondCell.style.color = "rgb(255,247,230)";  
    }  
    secondCell.style.fontSize = "2.5em";  
    thirdCell.innerHTML = sun + "%";***  
   A server test of this should give us the sun percentage for each day.
6. In your browser, open ***developer.forecast.io***. Go to the bottom and click the ***Terms of Service*** link. Under Attribution, you will see that there are some things that you must do to legally display data gotten from this service. This is an ***Intellectual Property*** prerogative. You must display a message and open a link to the site from your app. Let’s open ***solar.html*** and prep it for this task. After the closing ***</table>*** tag, enter the following code:  
    ***<p class="credit">Powered by <a   
    href="http://forecast.io">forecast</a></p>***Save that and copy it to your web domain.
7. Return to script.js and let’s enter the following code to finish this off. Make this the last line in ***fillWeather()***:  
    ***document.querySelector("section.week   
    p.credit").style.display = "block";***  
   Give this a server test and try out the link.