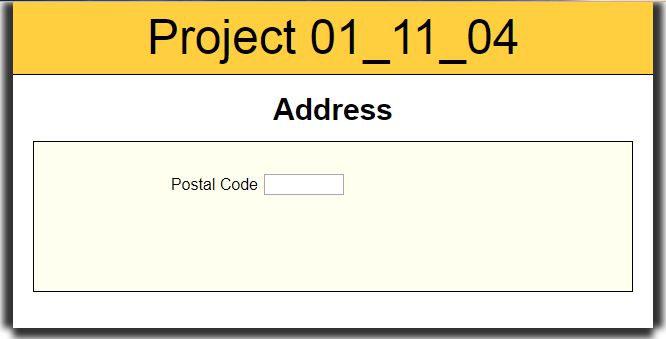
**Project 01\_11\_04**

In this project, you will enhance an address form using an AJAX request to autocomplete city and state fields based on a zip code entered by a user. You will use a web service to provide data. We will not need a proxy for the service, it enables CORS (Cross-Origin Resource Sharing) for all domains that use it.

1. Copy all of the completed files from ***Project 01\_11\_04.zip*** into a ***Project 01\_11\_04*** folder; this will contain all of the files you will zip up and submit. Open all of the necessary files in your IDE and complete the documentation at the top.
2. Go to the JavaScript file ***script.js***. At the bottom of the file, add the ***global*** variable ***httpRequest*** and set its value to ***false***:  
   ***var httpRequest = false;***
3. Let’s start to scaffold out some code that will set up our event listeners. This time we are interested in the ***keyup*** event. This will fire when the user releases a keyboard key in the ***Postal*** ***Code*** field. We will set its event handler to be ***checkInput()***. At the bottom of the file, create the following code:  
   ***var zip = document.getElementById("zip");  
   if (zip.addEventListener) {  
    zip.addEventListener("keyup", checkInput, false);  
   }   
   else if (zip.attachEvent) {  
    zip.attachEvent("onkeyup", checkInput);  
   }***
4. Now lets build just a shell of the ***checkInput()*** function so we can get a reasonable test. Add the function above the code we just did, we will build it out later:  
   ***function checkInput() {  
    alert("checkInput()");  
   }***Move that over to the server. Let’s test it by entering some data into the field with the keyboard. The ***alert()*** should fire when we release a key.
5. Now let’s build out the function to check the length of the zip code that was entered. If the length is correct we will call a ***getLocation()*** function. If not, we will remove the values from the city and state fields, which may not yet be visible. The following code should replace the debug ***alert()***:  
    ***var zip = document.getElementById("zip").value;  
    if (zip.length === 5) {  
    getLocation()  
    }  
    else {  
    document.getElementById("city").value = "";  
    document.getElementById("state").value = "";  
    }***  
   Let’s server test that. Open Developer Tools to the Console tab. After the fifth digit is entered, we should get a Console error. It tells us getLocation() is not defined, which is true.
6. Now let’s scaffold the ***getLocation()*** function and just fill it with a test, to make sure everything is okay. Let’s switch over to ***console.log()*** functions for debugging, just to get some practice. We can add the function shell just below ***checkInput()***:  
   ***function getLocation() {  
    console.log("getLocation()");  
   }***  
   Now lets server test and enter our 5 digits. We should get a message to the console if everything is okay.
7. We can now start to build out ***getLocation()*** by having it call the function ***getRequestObject()*** to either get a new ***XHR*** object or reuse an existing one. We will build that function to test the following code which will replace our ***console.log()*** debug:  
    ***var zip = document.getElementById("zip").value;  
    if (!httpRequest) {  
    httpRequest = getRequestObject();  
    }***  
   A server test of that tells us that we have to build our ***getRequestObject()*** function, which we will do next.
8. Directly above that, let’s create our ***getRequestObject()*** function. We will do that just below our global variables. We will then fill the function with a ***try*** / ***catch*** structure to create an ***XHR*** object. Add a console.log() that we will use to test the function:  
   ***function getRequestObject() {  
    try {  
    httpRequest = new XMLHttpRequest();  
    }  
    catch (requestError) {  
    return false;  
    }  
    console.log(httpRequest);  
    return httpRequest;  
   }***A server test should show us a valid ***XHR*** object in the console. We can now remove the ***console.log()*** debug statement.
9. Let’s flesh out the ***catch*** clause to do a little more work for us. If we can’t get an XHR object, we can still make the form useful. We can make the city and state fields visible, and remove the event listeners from the zip field. This will let a user complete the form manually without any AJAX help. Let’s add the following code just above the ***return*** in the catch clause:  
    ***document.getElementById("csset").style.visibility = "visible";  
    var zip = document.getElementById("zip").value;  
    if (zip.addEventListener) {  
    zip.removeEventListener("keyup", checkInput, false);  
    }   
    else if (zip.attachEvent) {  
    zip.detachEvent("onkeyup", checkInput);  
    }***
10. Now we have to figure out a clever way to test this out. We have to simulate not being able to get an ***XHR*** object. We can do that by misspelling the ***constructor*** as follows:  
     ***httpRequest = new ~~XML~~HttpRequest();***A server test should make our city and state fields appear. Additionally we should be able to add more digits to the zip or completely rewrite it and no AJAX request will happen. If that is the case, correct the spelling and test again.
11. We can now prepare our ***getLocation()*** function to generate our AJAX request. Add this code to the bottom of the function:  
     ***httpRequest.abort();  
     httpRequest.open("get", "http://api.zippopotam.us/us/" + zip,   
     true);  
     httpRequest.send(null);***A browser test with Developer Tools will be required to see some results. Go to the Network tab. We should see a Request URL that is properly formed.
12. Now we need to get our data, so we need to set our onreadystatechange event handler:  
     ***httpRequest.onreadystatechange = displayData;***A browser test here with the Console tab open should show us that we need to build our ***displayData()*** function.
13. Let’s start to build out the ***displayData()*** event handler below ***getLocation()***. First we will see if we can get data back, before we attempt to start placing that data on the page with the DOM. This data is going to come back in JSON format instead of plain text, so we will have to parse it:  
    ***function displayData() {***  
     ***if (httpRequest.readyState === 4 && httpRequest.status === 200) {  
     var resultData = JSON.parse(httpRequest.responseText);  
     console.log(resultData);  
     }  
    }***  
    A server test with Developer Tools in the Console tab should let us test. Make sure there are no syntax errors. Enter in a zip code you know, and we should get good ***console.log()*** data. Expand the object that is returned, then expand the ***places*** array. Expand the ***Object*** in it, and you should recognize data for that zip code.
14. Now let’s see if we can get some data onto the page by placing it into the DOM nodes. We will use the ***places*** array which is holding a nested array. It is ***associative***, so we can get its property values by name. Let’s replace the ***console.log()*** debug with the following code:  
     ***var city = document.getElementById("city");  
     var state = document.getElementById("state");  
     city.value = resultData.places[0]["place name"];  
     state.value = resultData.places[0]["state abbreviation"];  
     document.getElementById("zip").blur();  
     document.getElementById("csset").style.visibility = "visible";***A server test should give us our city and state data in the right place.
15. We would like to internationalize the project to handle German postal codes as well as United States postal codes. Go to the ***address.html*** file and let’s add a ***<fieldset>*** to implement some radio buttons to handle this. Directly below the opening ***<form>*** tag, let’s add the following HTML code:  
     ***<fieldset id="countryset">  
     <label for="germany" id="germanylabel">Germany</label>  
     <input id="germany" type="radio" value="germany"   
     name="country" />  
     <label for="us" id="uslabel">United States</label>  
     <input id="us" type="radio" value="us" name="country" />  
     </fieldset>***  
    Let’s do a server test to see how our modified form looks.
16. In script.js we should add a ***global*** variable to track the selected country. Directly below our global ***httpRequest*** variable, let’s add another one called ***countrySel*** with no initialization of its value  
    ***var countrySel;***
17. We are going to need a function to implement the radio buttons, much like we did a ***checkInput()*** function for the text boxes. Just before that function, let’s scaffold out the shell of a ***checkButtons()*** function, with an ***alert()*** in it so we can test:  
    ***function checkButtons() {  
     alert("checkButtons()");  
    }***
18. Let’s start to scaffold out some code that will set up our event listeners for the new function. This time we are interested in the ***click*** event. This will fire when the user checks one of the country radio buttons. We will set its event handler to be ***checkButtons()***. After the ***displayData()*** function, create the following code:  
    ***var germany = document.getElementById("germany");  
    var us = document.getElementById("us");  
    if (us.addEventListener) {  
     germany.addEventListener("click", checkButtons, false);  
     us.addEventListener("click", checkButtons, false);  
    }   
    else if (us.attachEvent) {  
     germany.attachEvent("onclick", checkButtons);  
     us.attachEvent("onclick", checkButtons);  
    }***Let’s give this a server test to make sure all is working. We should get an alert() indicating our ***checkButtons()*** handler is being called.
19. Now let’s build out our ***checkButtons()*** function to find out which button has been checked and to update our global ***countrySel*** variable. Let’s remove our debug ***alert()*** and add the following code:  
     ***var germany = document.getElementById("germany");  
     var us = document.getElementById("us");  
     if (germany.checked || us.checked) {  
     document.getElementById("zipset").style.visibility = "visible";  
     if (germany.checked) {  
     countrySel = "de";  
     }  
     else {  
     countrySel = "us";  
     }  
     }***  
    Let’s give this a server test to make sure there are no syntax errors and everything is still working.
20. Now let’s modify our ***httpRequest.open()*** function to make use of the ***countrySel*** variable so we get the correct data. Go to ***getLocation()*** and make the following modification to the code:  
     httpRequest.open("get", "http://api.zippopotam.us/***" + countrySel +"/"*** +   
     zip, true);  
    Definitely check this on the server for syntax. Also check the ***United States*** radio button and make sure the code still works for a zip code that you know. Now check the ***Germany*** radio button, enter ***80333***, and you should get the German city ***München***.
21. We need to do some cleanup now to make this operate correctly. We can start with some surgery on the ***catch*** statement in ***getRequestObject()***. We need to make sure we are handling our form field visibility correctly, as well as our event handlers. Go to the ***catch*** statement and make the following modifications:  
     catch (requestError) {  
     ***document.getElementById("zipset").style.visibility = "visible";***  
     document.getElementById("csset").style.visibility = "visible";  
     ***var germany = document.getElementById("germany");  
     var us = document.getElementById("us");***  
     var zip = document.getElementById("zip").value;  
     if (zip.addEventListener) {  
     ***germany.removeEventListener("click", checkButtons, false);  
     us.removeEventListener("click", checkButtons, false);***  
     zip.removeEventListener("keyup", checkInput, false);  
     } else if (zip.attachEvent) {  
     ***germany.detachEvent("onclick", checkButtons);  
     us.detachEvent("onclick", checkButtons);***  
     zip.detachEvent("onkeyup", checkInput);  
     }  
    Check this on the server for syntax and to see if it is all still working. You should not notice any differences yet.
22. In order to get the visibility of the fields working correctly, we are going to have to modify the ***CSS***. Go to the styles.css file and make the following ***selector*** change:  
    ***#zipset,*** #csset {  
     visibility: hidden;  
    }  
    A final server test should now start the ***City*** and ***State*** fields off as hidden.