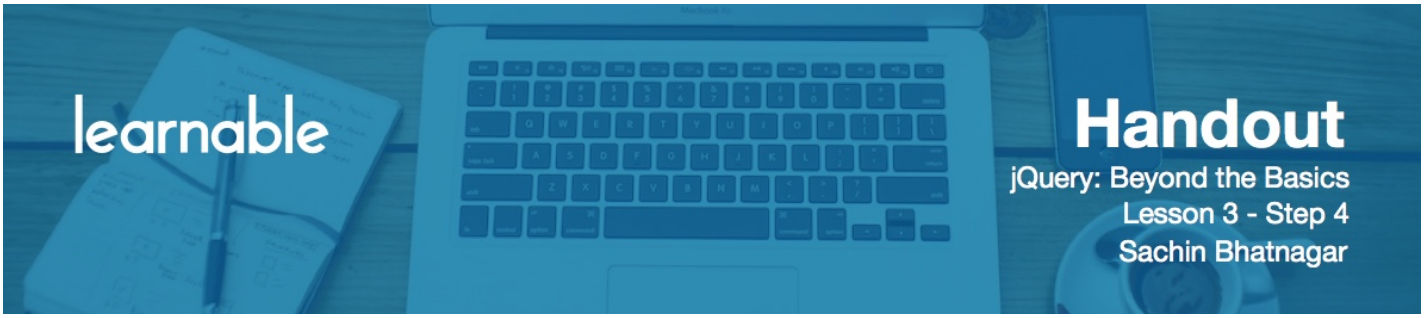

Introduction to APIs

Modern websites and mobile apps access and send data to all sorts of third party services. This is achieved through an **application programming interface (API)**. Your web app sends a request to a third party API and it responds back with data which your app then consumes. Web apps thus employ AJAX to access these APIs.

Basic process of using an API

You sign up with an API provider, let's say Twitter, and then get an **API key**, which is a sort of a password that secures your access. Then, whenever you place an AJAX call to the API, you pass in this key.

There is one very important thing to note here. Let's say the app that wants to use AJAX to access an API sits on one domain while the API itself is served from some other domain. In such case, your browser will not allow JavaScript to execute this AJAX request due to a **same origin policy** that browsers follow to prevent malicious requests from being sent out to third party servers.

AJAX requests across domains with JSONP and CORS

To resolve this issue there are two distinct techniques that developers follow. Let's discuss how these techniques really work. Please note that the same origin policy is not a very secure policy to prevent malicious attack on a server and there are other ways of enforcing security.

- **JSON with padding (JSONP)**. When you place an AJAX request you also pass in a parameter that mentions the name of a callback function, for example *http://somedomain.com/api/req.php?usr=John&callback=getData*. The code on the server is designed to return back the JSON response wrapped inside a function code:

```
getData({'user': 'John Doe', 'email': 'john@someapp.com'});
```

Back inside your JavaScript code this function can then accept the data. By wrapping the response JSON in a function call, the same origin policy is bypassed, allowing cross domain data interchange. As you can guess, implementing this requires a little extra effort at both the frontend and the backend.

- **Cross origin resource sharing (CORS)**. This technology involves the server sending back an extra header known as the **Access-Control-Allow-Origin header**. Once your browser receives this header it allows cross domain communication without any limitations and data comes in as regular JSON.

Comparing JSONP and CORS:

- JSONP is supported by virtually every browser on the planet, it even works in Internet Explorer versions below IE 7. CORS, on the other hand, is a newer feature and it works in most modern browsers. There is partial support in Internet Explorer 8 and 9 with full support starting from IE 10.
- Both these methods involve cooperation of the server. In JSONP the server has to wrap the JSON data within a function call. With CORS, the server needs to send the Access-Control-Allow-Origin header.
- JSONP is not as secure as CORS. If your project needs to target only modern browsers, prefer using CORS.
- JSONP supports only GET request and though there are ways of working around this issue, they require additional code at the server. CORS supports all kinds of HTTP requests from GET to DELETE.
- jQuery supports both JSONP and CORS, except that CORS is not supported in Internet Explorer versions 9 and below.

Creating a weather widget

Let's create a simple weather widget which displays the current weather for your location. This example will make use of the Google Maps Geocoding API and the Forecast.io API for fetching the location and the weather data.

The widget will first fetch user's current location by using HTML5's Geolocation API. Once we have the latitude and longitude of the user's location, we'll make two AJAX calls: one to the Google Maps Geocoding API for fetching the user's city and country, and the other to Forecast.io's API, using JSONP for fetching weather details.

When you load this page for the first time, your browser may ask for your permission to allow location data access for this page. Make sure you click on Allow. In the JavaScript section I've already built an immediately invoked function that first checks to see if your browser provides access to the HTML5 geolocation API. If yes, we use the `getCurrentPosition` method to fetch the latitude and longitude and pass this data to two functions. The first one is called `getCityName` - it queries the Google Maps Geocoding API using AJAX to fetch the city and country name from the latitude and longitude coordinates that we have.

To build this function we have to fetch the API key from Google that will allow us to access the Geocoding API. To do that, navigate to console.developers.google.com.

Creating a project in Google developer console

You will need an active Google account to proceed.

- While in the developer's console, click on the "Create Project" button and enter the name for your project.
- Click on "Create" - you will be taken to the project's dashboard.
- Click on the APIs link from the APIs and Auth section in the left. You'll see all the APIs that Google provides.
- Look for the Geocoding API and click on it.
- Enable access by setting the corresponding switch to "On".
- In the Quota section you can see that Google allows us to send up to 2 500 requests per day. If your app is going to be used by a lot of people, Google provides paid access with much higher limits on daily usage.
- Go into the Credentials section and click on "Create new Key" button in the "Public API access" section.
- Click on the "Server Key" button so that an API key is generated for accessing the Geocoding API.
- Copy this API key as you'll need it soon.

Working with jQuery AJAX and Geocoding API

Visit developers.google.com/maps/documentation/geocoding/ to get quick access to the Geocoding API's documentation. Look for the "Reverse geocoding" section.

Back to the code:

```
function getCityName(lat,long) {
  $.get("https://maps.googleapis.com/maps/api/geocode/json?latlng=" + lat + "," + long + "&key=YOUR_KEY", function(data) {
    $('#cityDisplay').text(data.results[6].formatted_address);
  });
}
```

Replace `YOUR_KEY` with an API key that was generated in the developers console a moment ago.

Working with jQuery AJAX and forecast.io API

Now we need to access Forecast.io's API.

- Go to developer.forecast.io.
- Click on the "Register" button and sign up to access the API dashboard.
- Copy the URL that will be provided in the dashboard - this is a preformatted example with your own API key.

On to the code:

```
function getWeather(lat,long) {
```

```
  status.text('Getting Current Weather...');
  $.get(("https://api.forecast.io/forecast/0fecca61a71bcca0bc4e5eea1cf2e002/" + lat + "," + long), {units:"si"}, function(data) {
    status.text('Done !');
    $('#currentTemp').text(data.currently.temperature);
    $('#summary').text(data.currently.summary);
    $('#cover').fadeOut(500);
  }, "jsonp");
}
```

`units:"si"` means that we want to get values in the international system of units (and temperature will be returned in degrees Celsius). If you don't this parameter, you'll get the temperature value in degrees Fahrenheit.

In the case of Forecast.io's API, we'll need to resort to JSONP and jQuery makes that process super easy: just add a string `"jsonp"` as the last argument after the callback function in the `$.get` method.

Go ahead and test that in your browser!