

## FarmData2 Onboarding Activity 06

### Web APIs Technology Spike

#### Introduction:

In the prior activities you have learned how to structure a page using some basic HTML functionality including form elements and tables (A02). You have used Vue.js and JavaScript to make a page dynamic by binding the content of some of its HTML elements to data in the Vue instance (A03). Most recently, you used JavaScript functions and Vue directives to respond to user events (e.g. button clicks, key presses, etc). The code executed in response to these events modified the data in the Vue instance and thus, via Vue's data binding, altered how the page was rendered (A04).

Thus, far all of the data that we have used has been hard coded into the HTML or JavaScript. In this activity you will learn about and gain experience using Application Programming Interfaces (APIs). These interfaces provide a way for JavaScript code in your page to request data from an external service. You'll practice by building a page that performs currency conversions using a publicly available API. Then ultimately in the next activity you'll use the FarmOS API to request real FarmData2 data for use in your Harvest Report spike.

It is not required, but if you would like a good general introduction to the idea of APIs and what they do you can watch the video *What is an API? Introduction to Application Programming Interfaces with Google Maps APIs!* With Katherine from BlondieBytes:

- <https://www.youtube.com/watch?v=T74OdSCBJfw> (9:27)
  - Note that since this video was produced Google has begun requiring an API Key (a way to authenticating the user) to access its services. So the examples won't run as they are shown. But this video still provides a good conceptual overview of APIs and how they work.

#### Getting Started:

1. Synchronize the main branch of your local and origin FarmData2 repositories with the upstream and merge any changes to main into your feature branch (refer to past Activities if necessary). List here any files in the main branch that were changed. If there have been no changes to the main branch indicate that instead.

--

#### Adding Another Sub-Tab:

2. Make sure you have your feature branch checked out. Add another new sub-tab named A05 to the FD2 Example tab. Have the contents of this new tab be provided by the file a05.html.



Make a copy of your a04.html file into the a05.html file. Don't forget to clear the Drupal cache when you are done. The result should be that you now have an A05 sub-tab that is (for now) identical to your A04 sub-tab. You'll be working on the A05 tab throughout this activity.


3. Commit your changes to your feature branch with a meaningful commit message and push it to your origin. Recall that this also updates your Draft Pull Request.

### An API Tool:

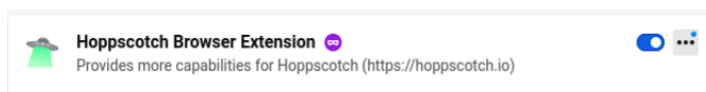
You'll use the *Hoppscotch application* to interact with and learn about APIs. Hoppscotch will allow you to manually send API requests and inspect the responses that are returned. Doing this manually is an excellent way to learn about APIs. But it is also an essential technique for designing and testing calls to APIs that you may want to use in an application. In this set of activities you'll use Hoppscotch to learn about a currency conversion API. But, eventually you will use it to design test calls to the FarmOS API which will provide access to the FarmData2 data that you need for your Harvest Report spike. You'll see that once an API call has been designed and tested using Hoppscotch, it is relatively easy to translate it into JavaScript code that makes the requests and processes the results.

4. Visit the Hoppscotch page: <https://hoppscotch.io>.

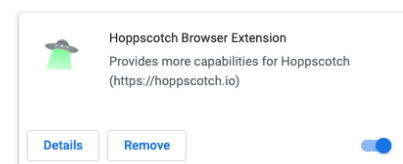
5. You'll learn more Hoppscotch in a minute, but first you'll need to install the Hoppscotch browser extension. To do so:

- Click on the  icon in the upper right.
- Choose Extensions
- Click on your browser "Firefox" or "Chrome"
  - This will take you to the Firefox Add-Ons site or the Chrome Web Store where you can install the extension.
- Click the "Add to..." button on the page that appears to install the extension.

6. Find the "Add-Ons" (Firefox) or "Extensions" (Chrome) in your browser and confirm that you have the Hoppscotch Browser Extension installed. Be sure that you see one of the boxes below depending upon your browser before continuing:



Firefox



Chrome

### The Exchange Rates API:



There is a lot to learn about API's before getting into the FarmOS (i.e. FarmData2) API. So you will start with the much simpler *Exchange Rates API* to learn the basics. The Exchange Rates API is a set of free *API endpoints* (i.e. URLs) published by the European Central Bank. Sending requests to these endpoints provides access to current and historical currency exchange rates.

7. The homepage for the Exchange Rates API (<https://exchangeratesapi.io/>) gives a number of examples of API calls for different purposes. Find the first example "*Get the latest foreign exchange reference rates*" under "*Usage*" on the Exchange Rates API page.

a. What is the URL of the API endpoint for the latest exchange rates? Note: The the GET and the HTTP/1.1 parts of the example are not part of the URL and should be omitted.

b. Put the URL for this API endpoint into the URL field in Hoppscotch and send it. What is the "Status" of the response in Hoppscotch (i.e. the green number)? If you do not see a green number labeled "Status" under "Response" on the Hoppscotch page make sure you correctly installed the browser extension and check the endpoint URL from part a.

c. The "Response Body" in Hoppscotch should now display information about the latest currency exchange rates. Paste that response body here.

The information in the "Response Body" is in *Java Script Object Notation (JSON)*. The JSON format should look reasonably familiar because it is the same format that is used to define JavaScript objects. If you'd like to review or learn more about JSON format you can find more information using the links below:

- *An introduction to JSON* by Raivat Shah:
  - <https://towardsdatascience.com/an-introduction-to-json-c9acb464f43e>
- *JavaScript Object Initializers* in the MDN pages:
  - *Object Initializer*: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Object\\_initializer](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Object_initializer)

8. Use the content of the "Response Body" in Hoppscotch to answer the following questions:



a. What are the names of the three properties in the JavaScript object in the response?  
Note: One of these three properties has an object as its value and thus contains more properties. This question is only asking for the three properties in the top-level object.

b. What is the name of the *property* whose *value* gives all of the exchange rates?

c. The value of the property "base" gives the symbol of the *base currency* for the conversions. What is the base currency for this response?

d. Use your favorite search engine to find the currency symbol for Thai Bhat.

e. What is the current exchange rate for the Thai Bhat with respect to the base currency in the response?

Some API endpoints can have additional information passed to them to customize their behavior. One way that this is done is to append a *Query Parameter* to the end of the URL for the API endpoint. This is done by following the API endpoint URL with a ? and then the information for the query parameter. The query parameter will be a string like: *name=value* where *name* and *value* are meaningful to the API.

9. Find the example on the Exchange Rates API (<https://exchangeratesapi.io/>) that allows you to make a request for a "Quote against a different currency".

a. What is the URL given for this example?



b. What is the query parameter given in this example?

c. Use your favorite search engine to find the currency symbol for Swiss Francs.

d. Give a URL including a query parameter requesting that the exchange rates should be quoted against the Swiss Franc.

e. Test your URL from part c using Hoppscotch. Examine the “Response Body that you receive. How can you tell from the response that the exchange rates are now quote against the Swiss Franc?

10. Find the example on the Exchange Rates API (<https://exchangeratesapi.io/>) that allows you to use currency symbols to “Request specific exchange rates”.

a. What is the URL given for this example?

b. What is the query parameter given in this example?

c. Give a URL including a query parameter requesting just the exchange rate for Swiss Francs quoted against the default base currency (i.e. the Euro - EUR.)



d. Test your URL from part c using Hoppscotch and paste the “Response Body” here.

11. Imagine that you want to know the rate for both Swiss Franc and for Thai Bhat at the same time. You could use two separate requests. But often API’s will allow multiple values to be given for a query parameter. For example using a string like: `name=value1,value2` where `value1` and `value2` are usually treated like a logical OR. That is, the response will include results for when `name=value1` or when `name=value2`.

a. Give a URL including a query parameter requesting the exchange rate for both Swiss Francs and Thai Bhat quoted against the default base currency (i.e. the Euro - EUR).

b. Test your URL from part c using Hoppscotch and paste the “Response Body” here.

12. Imagine you now want to obtain a quote for the exchange rate for converting Thai Bhat into Swiss Francs. You could do this by getting both rates in terms of the Euro and then doing a little math. But we can also do it by combining the two query parameters we have seen. To include multiple query parameters simply place an `&` between them giving a string like: `name1=value1&name2=value2` with the names and values being replaced with things that are meaningful to the API.

a. Give a URL including the query parameters to request the exchange rate for Swiss Francs using the Thai Bhat as the base currency.

b. Test your URL from part c using Hoppscotch and paste the “Response Body” here.



13. Often services will provide multiple different endpoints (i.e. URLs) with each providing different types of information.

- a. What is the endpoint URL that you would use to request historical exchange rates between two dates rather than the latest exchange rates?

- b. Give a URL including all query parameters requesting the exchange rates for Swiss Francs and Thai Bhat between May 1 and May 3, 2020 using Mexican Pesos (MXN) as the base currency.

- c. Test your URL from part c using Hoppscotch and paste the “Response Body” here.

#### **A Necessary Browser Extension:**

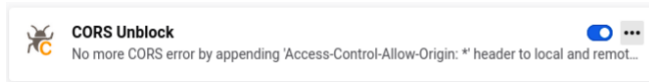
Now that you have a feel for how to make API requests using endpoint URL and query parameters the next step is to built that into a page using JavaScript. Just one thing before we can get to that. In order to write JavaScript code to use the Currency Exchange APIs you will need to install another extension in your browser.

14. Install the CORS Unblock extension in your browser using the appropriate link below.

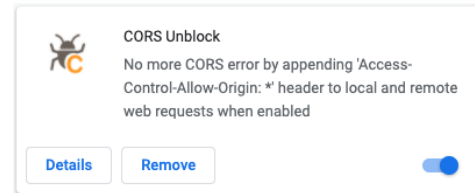
- For Firefox:
  - <https://addons.mozilla.org/en-US/firefox/addon/cors-unblock/>
- For Chrome:
  - <https://chrome.google.com/webstore/detail/cors-unblock/lfhmikememgdcahcdlaciloanbhhjino/>

15. Find the “Add-Ons” (Firefox) or “Extensions” (Chrome) in your browser and confirm that you have the CORS Unblock Browser Extension installed. Be sure that you see one of the boxes below depending upon your browser before continuing:





Firefox



Chrome

This extension is needed because most web browsers will only allow a page to make API request to endpoints in the same domain (i.e. web-site) from which it came. That is, if a page comes from xyz.com, then it will only be allowed to make API requests to endpoints at xyx.com. Call by a page from one domain to an API in another domain are called Cross Origin Requests (CORS). For security reasons browsers typically prevent CORS unless they are explicitly allowed by the API. The Hoppscotch application did some magic for us that allowed us to make CORS to the Exchange Rates API. The CORS Unblock browser extension will allow us to do the same from JavaScript code.

Note that the CORS Unblock browser extension can sometimes interfere with the operation of other web sites. So, it is recommended that you turn it off (click the blue slider) when you are not working on this assignment. If you start seeing CORS errors when you come back to the assignment that means you need to turn it back on.

### API Requests using Axios:

*Axios* is a JavaScript library that provides convenient methods for making API requests. In this section you'll learn about the essential aspects of Axios and use it along with Vue.js to build a simple currency converter page. Everything you need should be here, but if you'd like another resource at a similar level of detail you can check out *Using Axios to Consume APIs*:

- <https://vuejs.org/v2/cookbook/using-axios-to-consume-apis.html>

16. Create a new text file named `axiosspike.html` that is not in the the FarmData2 repo and paste in the starter contents below:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Axios API Spike</title>
  </head>
  <body>
    <h1>Currency Converter</h1>

    <div id="axiosspike" v-cloak>
      <p>Convert from
        <select v-model='fromCurrency'>
          <option v-for='currency in currencies'>{{ currency }}</option>
        </select>
      </p>
    </div>
```





```
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
<script src="https://unpkg.com/vue"></script>

<script>
  let spike = new Vue ({
    el: '#axiosspike',
    data: {
      fromCurrency: null,
      currencies: ['EUR', 'CHF', 'THB', 'USD'],
    },
  });
</script>
</body>
</html>
```

17. Load your `axiosspike.html` file into your browser and answer the following questions.

a. What line includes the Axios library?

b. What options appear in the drop-down list when the page is loaded?

c. Briefly explain why those specific options appear in the list.

d. Which property in the Vue instance is bound to the selected currency? You can use the Vue devtools to confirm this binding.

18. Now instead of hard coding the list of currencies that are available it would be better to get this information from the Currency Exchange API and use that to populate the drop-down list. We can do this using a combination of Axios and Vue.js. We will build this up step by step so that you can see how each piece works.

a. Change the hard coded value of the `currencies` property of the Vue instance to be an empty array (i.e. `[]`). Reload the page to confirm that the drop down is now empty.



b. Now we will want to populate the `currencies` array with data when the page is loaded (i.e. when the Vue instance is created). Conveniently, if we add a function named `created` to our Vue instance it will be called automatically when the Vue instance is created. Add the following `created` function to your Vue instance below the `data` property:

```
created() {  
  this.currencies = ['EUR', 'CHF', 'THB', 'USD'];  
},
```

Reload the page to confirm that the drop down now contains the currencies again.

Because the currencies are still listed explicitly it may seem like we haven't done anything at all. But we have! The `currencies` are now set by code in the `created` function that is executed when the Vue instance is created. All we'll need to do is now replace that code with an API call.

c. Replace the statement in the body of the `created` function with the following call to the Axios get method:

```
axios.get('https://api.exchangeratesapi.io/latest')  
  .then(response => {  
    this.currencies = response.data.rates;  
  })  
);
```

Reload the page and check the contents of the dropdown. It probably won't be what you were expecting. But we'll fix that in a moment. For now, describe what was contained in the dropdown and how it relates to the Response Body you saw in question #7c.



So, what happened... The above code makes an API call to the endpoint specified in the call to the `axios.get` method. When the response is received the `.then` method is invoked with the `response` and any statements in the `{ }` are executed.

In this case, the statement:

```
this.currencies = response.data.rates;
```

is executed. That statement sets the `currencies` property in the Vue instance to be an array of the values that are contained in the `rates` object of the response. This is what JavaScript does by default, but unfortunately in this case that's not what we wanted. We wanted the names of the properties not their values. We'll figure out how fix that in a moment.



Before we do, notice that we have to use `response.data.rates` to get to the `rates` object from the “Response Body” because Axios actually returns a *response object* with a lot of extra information and it places the “Response Body” we are interested in in the `data` property within that object. For now, you’ll just need to remember that when you want to access the “Response Body” you’ll need to use something like `response.data.xyz` where `xyz` the name of the property you want from the “Response Body”.

19. How would you access the `base` property in the Axios response if you wanted to use it in your application?

So now, let’s get the drop-down to display the currency symbols instead of the exchange rates. To do so, we need to set `currencies` to the property names of the `rates` object not their values, which is what JavaScript does by default. So, what we need is an array of the property names that exist in the `rates` object. There are a number of different ways to do this in JavaScript. But before we find one, if you have worked with maps or dictionaries in other languages (e.g. A `HashMap` in Java or a `Dictionary` in Python), this should sound very similar to getting the keys rather than the values.

20. Using your favorite search engine, search for something like *JavaScript get array of property names*. Review a few of the results and study the examples they contain. Use what you find to modify the code in your spike so that the drop-down displays the currency symbols. Hint: There are several good solutions that only require you to modify the one line in the `.then` part of the axios statement. If you find yourself making big changes or adding lots of code, step back and review the search results again.

Paste the line that you changed here.

21. This is all great if the Currency Exchange API works and the request is properly formed. But in any networked system there is a chance that something will go wrong. To simulate a problem where the server could not be found, modify the URL of the endpoint for the Currency Exchange API in your `created` method by adding a few random characters to the end (e.g. `latestxyz`).

a. What is in the drop-down when the page is reloaded?



b. Use the devtools console to find out what went wrong. What error is reported?

To account for situations like this we can add a `.catch` to the axios statement. A `.catch` lets us define a block of statements that will be executed if the API request fails for any reason. The general form of an axios statement including `.get`, `.then` and `.catch` looks like the following:

```
axios.get('endpoint url')
  .then(response => {
    statements to run if API call fails.
  })
  .catch(error => {
    statements to run if API request fails.
    console.log(error);
  })
);
```

Note that the `console.log(error);` statement in the `.catch` is not strictly required. However, it prints any JavaScript error messages to the devtools console, which can be very helpful in debugging. So, it is recommended that you include it in all `.catch` blocks.

21. Add a `.catch` clause to the axios statement in your created method so that if the API call fails the drop-down for the currency the page will contain the message “Service Unavailable” instead of the list of currencies. Not the best error reporting for a UI/UX perspective, but okay for now. Paste your `.catch` statement below.

### **Completing the Currency Converter Spike:**

You now have all of the basics necessary to fill out the functionality of the Currency Converter.

22. Add HTML to create a second dropdown for specifying the currency to convert to. For example:



### Currency Converter

Convert from CAD ▼

Convert to CAD ▼

Be sure to bind the value of this drop-down to a property in your Vue instance.

a. Paste the HTML that you added here.

b. Paste the property that you added to your Vue instance here.

23. Add a button that will (eventually) cause exchange to be requested and displayed. For now, so that we don't add too much new code at once just have the button call a method that causes the displayed result to be incremented by 1 each time the button is clicked. Be sure to make use of Vue data binding and the double mustache to display the result. For example, if the "Get Rate" button had been clicked 5 times you might see:

### Currency Converter

Convert from HUF ▼

Convert to RON ▼

Get Rate

Result: 1 HUF is 5 RON

a. Paste the HTML that you added here.

b. Paste the `methods` property of your Vue instance including the function that is called when the "Get Rate" button is clicked here.



24. Now let's get and display the actual exchange rate! Modify the code in the method called when the "Get Rate" button is clicked so that it uses Axios to request the appropriate exchange rate. Hint1: Use the specified currencies to Hint2: Start by putting `response.data.rates` into the property in your Vue instance that is bound to the displayed results. You'll notice that it is not quite what you want. For example:

### Currency Converter

Convert from DKK ▼

Convert to AUD ▼

Get Rate

Result: 1 DKK is { "AUD": 0.2067880483 } AUD

Paste the `methods` property of your Vue instance including the function that is called when the "Get Rate" button is clicked here.

If you think about it the above result should be a little surprising. When we put `response.data.rates` into the drop-down earlier we just saw the values and not the properties. But now we are seeing the whole object. This type of quirky inconsistency is something you will run into frequently when working with HTML, Vue and JavaScript. When it happens, the best thing to do is to figure out a work around. And the best way to do that is often by doing a well-crafted web search.

What we need to here is to use the symbol for the target currency (e.g. AUD) to get the associated value (e.g. 0.2067...) from the `rates` object. Said another way, we need to use the property name to get its value.

25. Using your favorite search engine, search for something like *JavaScript get property value by name*. Review a few of the results and study the examples they contain. Use what you find to modify the code in your spike so that just the exchange rate is displayed. Hint: There are several good solutions that only require you to modify the one line in the `.then` part of the axios statement. If you find yourself making big changes or adding lots of code, step back and review the search results again.

Paste the line that you changed here.



26. You may have noticed that the currencies in the drop-downs are not in any particular order. That would make it difficult for a user to find a specific currency that they are looking for. It would be better if they were in alphabetical order. Using your favorite search engine figure out how to sort an array in JavaScript and modify your spike so that the currency lists appear in alphabetical order. Hint: This can be done by adding one line to the `axios .then` block in the created function. If you find yourself making big changes or adding lots of code, step back and review the search results again.

Paste the line that you changed here.

### **Adding to the Harvest Report Spike 1:**

The FarmOS API is quite complicated so we aren't quite ready to fully complete our Harvest Report spike. However, the following activities will move in that direction and give you a little feel for the what's coming in the next set of Activities,

27. Be sure that FarmData2 is up and running. Open a browser tab and log into FarmData2 as a manager or worker.

28. In the same browser open a tab and go to Hoppscotch (<https://hoppscotch.io>) and put in the following API request:

```
http://localhost/taxonomy_term.json?area_type=field
```

The "Response Body" will have a property named `list`, which is an array of objects. Each of the objects in `list` describes one of the fields on the farm. The object for each field gives a lot of information about that field. For now however, we will only be interested in the names of the fields so that we can populate the drop-down in the Harvest Report Spike. Which property in each object gives the name of the field?

29. Add a `created` method to your Harvest Report spike on the A05 tab that requests the list of fields using the FarmOS API and uses it to populate the drop-down. To make this work correctly there are a few other things you'll need to know:

- Because you are issuing the request from the same origin as the page you will not include the full URL of the API request in your `axios .get` statement. Instead of the full request you used in question #28, leave off the hostname, and just use:



```
/taxonomy_term.json?area_type=field
```

- As you saw in question #28, the response contains a lot more than just the field names. But to populate the drop-down you'll need an array containing just the field names. To create this new array we need to go through all of the objects in the `list`, getting the `name` property of each and adding it to a new array. There are lots of ways to do this. But the JavaScript `map` function provides a particularly convenient syntax for this type of operation. For example, the the following statement will assign `fields` in the Vue instance to be an array containing all of the field names:

```
this.fields = response.data.list.map(f => f.name);
```

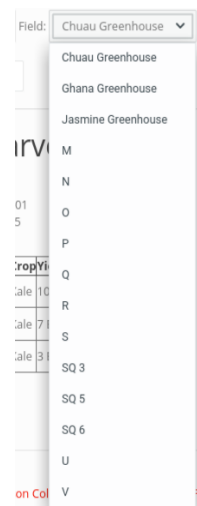
The `map` function, sort of like a `for-each` loop, assigns `f` to be each element in the array `response.data.list` in turn. Then for each of those elements, `f`, the arrow operator (`=>`) performs a specified operation (e.g. `f.name`) on it and places the result into a new array. The end result in this case is an array containing the value from the `name` property from each object in `list`, which is exactly what we want.

If you want another explanation with a little more detail the first part of article *Simplify your JavaScript – Use `.map()`, `.reduce()`, and `.filter()`* from the poka-techblog gives a nice overview:

- <https://medium.com/poka-techblog/simplify-your-javascript-use-map-reduce-and-filter-bd02c593cc2d>

- The list of fields should appear in alphabetical order.

Reload your page and ensure that the drop-down for the fields now contains a complete list all of the fields on the farm. For example, see the image at the right.



30. Commit your changes to your feature branch with a meaningful commit message and push it to your origin.

### **Adding to the Harvest Report Spike 2:**

31. Add another axios request to your created method to populate the list of crops using an the API request :

```
/farm_asset.json?type=planting
```

Hint: The response to this request will have a list with one object for each crop. Use Hoppscotch to find the property that has the name of the crop and then adapt what you did in question #29 to get an array of just the crop names.





Reload your page and ensure that the drop-down for the crops now contains a complete list all of the crops that can be planted. There are a lot of them and you may notice that a few of them appear multiple times. This is a recently discovered issue with the contents of the database that will need to be resolved in the future. So no need to worry about it.

32. Commit your changes to your feature branch with a meaningful commit message and push it to your origin.

### **Currency Converter UI/UX Improvements:**

This section is optional. You may have noticed some quirky behavior when playing around with your currency converter spike. For example, you can click “Get Rate” without having one or the other or both of the currency drop-down’s set. Also, when you change one of the drop-downs the currency reported in the result changes but the actual conversion rate does not change. These are poor User Interface (UI) / User Experience (UX) design choices and would need to be improved in any product actually used by real people. Here are some improvements that you can tinker with if you’d like to improve it:

- Have the “Get Rate” button be disabled unless both currencies are set.
- The Euro is missing from the lists! This is because by default it is the base currency and is not included in the rates object. Add the Euro (EUR) to the currency lists.
- The error reporting by setting the value of the drop-down is not very effective. If the service is unavailable, have the entire form hidden and display a message to that effect.
  - Improve this by adding a button to “Check Again” if the service is available now.
- Change it so that incorrect results are not displayed (e.g. one of the currencies is changed after “Get Rate” is clicked). There are a number of ways to do this:
  - The easiest is to request a new conversion every time a currency is changed. This works but will generate more network traffic than necessary.
  - A more efficient, but more difficult solution, is to save a response that has all of the rates for one base currency. Then use a computed property so that anytime there is a change the new rate is computed. This will require a little arithmetic to covert to the base currency and then to the desired currency. More work, but once it’s done it will be a very elegant solution.

