

FarmData2 Onboarding Activity 07

FarmOS API Technology Spike

Introduction:

At the end of activity A06 you were able to use Axios to retrieve some data from FarmData2 through the farmOS API. This activity will focus on learning more about the structure and basics of data in farmOS and the farmOS API. You won't learn everything but hopefully enough so that when you need additional or different data from FarmData2 you'll know where to look and how to experiment with FarmData2 and the farmOS API to find what you need.

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 A06 to the FD2 Example tab. Have the contents of this new tab be provided by the file a06.html. Make a copy of your a05.html file into the a06.html file. Don't forget to clear the Drupal cache when you are done. The result should be that you now have an A06 sub-tab that is (for now) identical to your A05 sub-tab. You'll be working on the A06 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.

The farmOS Data Architecture and Interface:

The Architecture page (<https://farmos.org/development/architecture/>) provides a high-level overview of the key *entity types* that farmOS, and thus FarmData2, uses to organize its data. If you are familiar with object-oriented (OO) programming, you can think of entity types as being classes. Similarly, specific instances of entity types are just called *entities*, and these correspond to objects in OO programming.

4. What are the four primary *entity types* used by farmOS? For each one, give its name, a brief description of what it represents and a few examples of the types of things that it is used for.



5. Which of those entity types appear directly as items in the menu at the top of the FarmData2 interface?

There is also a video at the top of that page where Matt Stenta, the creator of farmOS, is interviewed by Chris Callahan about farmOS's architecture and gives a little walk-through of the project. This video is worth viewing as it provides some additional important context. The video is also available directly from YouTube:

- *FarmOS Tutorial: Structure and Architecture Overview*
 - https://youtu.be/1wXD_K7Y_al

6. In the video Matt describes all four of the items that appear in the menu at the top of the FarmData2 interface and relates them to the terms: who, what, where and when. How does Matt relate each of the four menu items to one of these terms?

7. Log entities play an essential role in the organization of farmOS data. Summarize in your own words how Matt and Chris describe the role of logs in relationship to the other three types of entities (mentioned several times: between 8:00-9:00 and again near the end).

8. Use the farmOS interface to answer the following questions.

a. What type of assets exist in FarmData2?

b. What types of logs exist in FarmData2?



c. Who are the people in the development instance of FarmData2 and what are their roles?

farmOS API Endpoints:

As you saw at the end of A05, farmOS provides an API for accessing its data. The *API* page (<https://farmos.org/development/api/>) provides the documentation for this API.

The page begins with information on authentication. Because tabs in FarmData2 run from within farmOS and users must be logged in to use them, we will not need to use authentication. So, you can safely skip over those sections to the *API Version* section of the page. The early examples that use the `curl` command here show how to make farmOS API requests using the command line tool `curl`. The `curl` tool is useful, but again will not be particularly relevant to us. Instead, you will continue using Hoppscotch to explore and experiment with the farmOS API.

9. To get setup for the following activities you will need to:

- Ensure that FarmData2 is up and running.
- Open a browser and connect to FarmData2.
- Log in to FarmData2 as a worker or manager.
- Open another tab in the same browser and load the Hoppscotch site.
- <https://hoppscotch.io>

The information relevant to us begins with the *Endpoints* sub-section. Notice that the list of endpoints provided is divided into three groups. Those groups roughly correspond to three of the four entity types in farmOS and two of them correspond directly items in the menu at the top of the FarmData2 interface.

10. Compare the endpoints listed for Assets and Logs to the menu options in FarmData2. What do you notice?

11. Notice the ... in the list of endpoints. This indicates that there are more endpoints that exist but that are not listed.

a. What are some additional endpoints that you think would exist for Assets?



b. What are some additional endpoints that you think would exist for Logs?

The close relationship between the FarmData2 menu items and the API endpoints provides a convenient way to figure out where the data that you might need for new FarmData2 features can be found.

12. For example, imagine we want to find all harvests from a particular field. Use the FarmData2 interface to find all harvest logs from the field Jasmine-1. Hint: Use the “Filters” tab once you go to the harvest logs.

a. How many harvest records are there for Jasmine-1?

b. What crops were harvested from Jasmine-1?

c. On what dates did the harvests occur?

13. To access the same information using the farmOS API we will need to use one of the endpoints.

a. Give the URL including the query parameter of the endpoint that we would use to access harvest data through the API. Note: Do not worry about specifying the field yet.



b. Use Hoppscotch to make a request from the URL you found in part a. You should get a response that contains a list of harvest logs. If not, revisit part a and check the syntax of your request in Hoppscotch. Notice that the harvest logs in FarmData contain a lot more information than those we've been working with in our spike so far.

c. Each log in FarmData2 has a unique identifier with the property name `id`. What is the value of the `id` property of the first harvest log in the `list`?

d. What is the value of the `name` property of the first harvest log in the `list`.

e. From which field was that crop harvested? Hint: Look through the first harvest log for the name of one of the fields on the farm. You can find the list of fields using the FarmData2 interface or in the drop-down list from your A05 sub-tab if you don't remember them.

f. Notice that many of the properties in the harvest log have arrays and/or objects as their values. For example, find property named `uid` and notice that its value is an object (note the `{ }`), which has a few properties of its own (e.g. `uri`, `id`, `resource`). Similarly, the value of the `quantity` property is an array with an object (note the `[{ }]`) inside of it. The field name that you found in part e is also inside of an object that is inside of an array.

Copy the property name and its value (i.e. the full array and object) that contains the field name here.

Filtering farmOS API Results:

We don't always want all of the results that a farmOS API endpoint will give us. For example, we might just want all of the harvest logs from a particular field, all of the harvests of a particular crop, or all of the harvests between two dates. The farmOS API will allow us to filter the results using query parameters. For example, the request you used above has already done this.



14. Look closely at the URL that you used in 13a, which should have been:

`http://localhost/log.json?type=farm_harvest`

a. What is the query parameter in this URL?

b. What is the value of the query parameter in this URL?

c. Look at a number of the objects in the response from the farmOS API.

i. Do they all contain a property named type?

ii. What is the value of the type property in each of them?

iii. Why do you think the value of the type properties are all the same?

15. You can filter on additional properties by adding more query parameters to the request. Recall that you separate multiple query parameters using an &.

a. Give a URL that requests only the harvest log with the id that you found in for a request that using the id of the first harvest log you found in question #13c. Be sure to test your request using Hoppscotch. The response should contain only one harvest log, the one with the id that you specify.



b. Give a URL that requests that same harvest log but this time using its name. Hint: You might be tempted to put quotes in your URL because the name has spaces in it, don't do it! Be sure to test your request using Hoppscotch. Again, when you have this right the response that you receive should contain only one harvest log, the one with the name that you specify.

16. When the values we want to filter on are embedded within a property the farmOS API can (sometimes) be fairly clever. For example, the `uid` property described earlier (see #13f) contains information about the FarmData2 user that created this harvest log including the `id` property that identifies that user. If we wanted to find all harvest logs created by the user with the `id` of 4 we would make a request like:

```
http://localhost//log.json?type=farm_harvest&uid=4
```

a. How many harvest logs were created by the user with `id` 4?

b. How many harvest logs are returned if you use the nested property `id` in the query parameter instead of the top-level property `uid` by mistake (i.e. if you use the query parameter `id=4` instead of `uid=4`)? Why would that be? Hint: See #15a.

17. Give a URL that will request for all harvests from the Jasmine-1 field. Be sure to test your request using Hoppscotch. The response should match what you found in question #12.

Note that I said earlier, the farmOS API can *sometimes* be very clever when you use a top-level property in a query parameter. I said sometimes because the depending on the property the API may or may not look inside the object. There is no good documentation on which properties work which way. So, just be prepared to try some things out and see what works!

FarmData2 Dates:



A lot of the reporting and logging features of FarmData2 will involve the use of dates. For example, in your FarmData2 spike, the mock harvest report allows you to specify the start and end date for the report. Similarly, in FarmData2 every harvest log that is created will have a date associated with it. Thus, we will need to understand how FarmData2 handles dates.

18. Dates in FarmData2 are stored using *timestamps*. These timestamps are integer numbers that represents dates using *Unix time*. Using your favorite search engine, read a little about Unix time.

a. In a sentence or two of your own words describe how an integer timestamp represents a date.

b. What date and time does the timestamp 0 correspond to in Universal Coordinated Time (UTC)?

c. What is your time zone relative to UTC (e.g. UTC+5 or UTC-7, etc)?

d. What date and time would your answer to part b translate to in your time zone?

20. Each harvest log contains three timestamps: `timestamp`, `created` and `changed`. Read about each of these timestamp fields in the farmOS API documentation from earlier (<https://farmos.org/development/api>). You can find documentation of “standard fields” for most log types under the *Creating Logs* heading. Which of these three timestamps represents the time at which the harvest occurred?

21. Using the results of your request from question #17, what are the timestamps corresponding to the dates when the three harvests from Jasmine-1 occurred?



22. Give a URL that will request just the first harvest report from the Jasmine-1 field by using the timestamp for the date on which it occurred. Be sure to test your request using Hoppscotch.

Working with Dates in JavaScript:

While FarmData2 and computers in general are happy to work with dates using integer timestamps, they are not very convenient for us humans. So, we'll need to be able to convert back and forth between timestamps and more user-friendly ways of displaying dates. Learning to do this is a good task for a spike.

23. Create a new file named `datespike.html` outside of the FarmData2 repository and paste in the content below.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Unix Time Stamp / JavaScript Date Spike</title>
  </head>
  <body>
    <div id="datespike">
      <h1>Unix time stamp to JavaScript Date</h1>
      <label for="timestamp">Unix Time</label>
      <input type="number" id="timestamp" v-model='unixtime'><br>
      is {{ timestampToDate(unixtime) }}
    </div>

    <script src="https://unpkg.com/vue"></script>

    <script>
      let spike = new Vue ({
        el: '#datespike',
        data: {
          unixtime: 0,
        },
        methods: {
          timestampToDate: function(timestamp) {
            return "I don't know";
          },
        },
      });
    </script>
  </body>
</html>
```



```
    </script>
  </body>
</html>
```

24. Open your `datespike.html` page in a browser and answer the following questions based upon the above code and what you see in the browser. Your answers should be specific about how what you are seeing is related to the Vue instance.

a. Why does the timestamp field display the value 0?

b. Why does the text following the timestamp field display “I don’t know”

25. Modify the `timestampToDate` function so that it returns a JavaScript Date object for the provided timestamp. Use your favorite search engine to figure out how to do this. A search like “Unix timestamp to JavaScript Date” should get you going.

a. Paste your modified `timestampToDate` function here. Hint: This function should contain a single line of code.

b. What date and time are displayed when the timestamp field holds 0? Note that the Date object automatically adjusts to the local time zone. So, what you see should agree with your answer to #18d. If not revisit your `timestampToDate` function and question #18 to resolve the discrepancy.

c. Give the date strings that are displayed when you use your converter to convert the timestamps from question #21 into dates. These should agree with the dates you found in question #12c. If not revisit your `timestampToDate` function and your answer to #12c to resolve the discrepancy.



26. Eventually we will need to convert date to and from the YYYY-MM-DD format that is required by the date input element. With some searching you could figure out how to do these things. To save you some time, just add the following two functions into the methods element of your Vue instance.

```
timestampToYMD: function(timestamp) {
  let d = this.timestampToDate(timestamp);
  const month = d.getMonth() < 9 ?
    '0' + (d.getMonth() + 1) :
    d.getMonth() + 1;
  const day = d.getDate() < 10 ?
    '0' + d.getDate() :
    d.getDate();
  return d.getFullYear() + "-" + month + "-" + day;
},
YMDToTimestamp: function(ymd) {
  let d = new Date(ymd);
  return Math.round(d.getTime()/1000 +
    new Date().getTimezoneOffset()*60);
},
```

If you are curious about how some of this works you can find more information about the JavaScript Date object and its methods on the MDN pages:

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date

27. Let's add a conversion from the time stamp to the YYYY-MM-DD format first. Add some HTML code to your page so that it will now display the date in YYYY_MM_DD format as well. Hint: Use one of the above functions! When complete your page should look something like:

Unix Timestamp to JavaScript Date

Unix Time
is Wed Dec 31 1969 19:00:00 GMT-0500 (Eastern Standard Time),
or just 1969-12-31.

Paste the HTML that you added here:



28. Now let's add a conversion to from a date input element to a timestamp. When complete your page should look something like:

Unix Timestamp to JavaScript Date

Unix Time
is Thu Jan 01 1970 00:00:00 GMT-0500 (Eastern Standard Time),
or just 1970-01-01.

JavaScript Date to Unix Timestamp

Date: ☐ is 18000 as a time stamp.

For this one you will need to add some HTML code for the date element and the result as well as a new data field in the Vue instance for the data binding. Paste the HTML that you added here:

Dealing with Date Ranges:

When doing a search for a range of dates we will need to be able to find all of the timestamps that fall between the start date and the end date. It will be convenient for us to think about our search starting from the beginning of the day on the start date and going to up to but not including the beginning of the day after the end date. For example, if we are searching for harvests from July 1, 2018 through July 10, 2018 then we will want to find all records from 00:00am on July 1, 2018 up to but not including 00:00am on July 11, 2018.

Conveniently the `YMDToTimestamp` function gives us the timestamp for 00:00am on the specified date. Given that timestamp, we can get the time stamp for 00:00am on the following day simply by adding the number of seconds in a day.

29. Add HTML and a new function so that your page will now also display the timestamp of the day following the one chosen on in the date element. Your page might then look like:




Unix Timestamp to JavaScript Date

Unix Time

is Mon Jul 02 2018 01:00:00 GMT-0400 (Eastern Daylight Time),
or just 2018-07-02.

JavaScript Date to Unix Timestamp

Date  is 1530421200 as a time stamp.

The timestamp of the following day is 1530507600

a. Paste the HTML that you added here.

b. Paste the new function that you added here.

Advanced Query Parameters:

Now in order to be able to request harvest logs for a range of dates we need to be able to query for all of the logs with a time stamp greater than or equal to the time stamp of the start date and less than the time stamp of the end date. This is possible in farmOS API by augmenting our query parameters with operations.

These operations include:

- [lt] numerically less than
- [le] numerically less than or equal to
- [gt] numerically greater than
- [ge] numerically greater than or equal to
- [ne] numerically not equal to
- [eq] numerically equal to
- [sw] string starts with
- [ct] string contains

So, for example to request all of the harvest logs created by users with an id greater than 3 we could use the URL:

`http://localhost//log.json?type=farm_harvest&uid[gt]=3`



We can also include multiple query parameters. For example, to request all of the harvest logs created by users with `id 2` or `id 4` we could use the URL:

```
http://localhost//log.json?type=farm_harvest&uid[eq]=2&uid[eq]=4
```

30. Give URLs for the following requests. For each one, issue the request using Hoppscotch and then check your results using the FarmData2 interface.

a. All harvest logs from the fields Jasmine-1 or Jasmine-3.

b. All harvest logs from the field Jasmine-1 that occurred on or before August 9th 2018.

c. All harvest logs from the field Jasmine-3 that occurred between March 28th 2018 and September 24th 2018.

Adding to the Harvest Report Spike 1:

The next several sections will slowly build up until some of the search functionality of your mock harvest report is working. **It is not necessary that you complete all of the following questions. Attempt each in turn and see how far you can get.**

31. Modify the code in your A06 sub-tab so that when the Generate Report button is clicked a request is made for all of the harvest logs from the Jasmine-1 field. Use a map statement like the following in the `.then` to process the full FarmData2 harvest logs in the response down to our simplified harvest log records:

```
.then(response => {
  this.harvestLogs = response.data.list.map(h => {
    return {
      date:h.timestamp,
      field:'blah',
      crop:'blah',
      yield:'blah',
    };
  });
});
```



})

Note that the map used here is similar to the other ones we have used but it returns an object. This will populate the harvest reports with rows that look like:

Date	Field	Crop	Yield	Delete
1531368000	blah	blah	blah	Delete
1533787200	blah	blah	blah	Delete
1535342400	blah	blah	blah	Delete

The date in each row is the timestamp from the harvest log and the other properties were filled with the string literal 'blah'.

32. 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:

33. Modify the code in your A06 sub-tab so that the dates in the Harvest Report are converted from time stamps to human readable form.

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

Adding to the Harvest Report Spike 3:

35. Modify the code in your A06 sub-tab so that the Field, Crop and Yield cells in the Harvest Report have the correct values from their associated FarmData2 harvest logs.

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

Adding to the Harvest Report Spike 4:

37. Modify the code in your A06 sub-tab so that harvest report that is generated is for the field that is selected in the Field drop-down menu.

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

Adding to the Harvest Report Spike 5:



39. Modify the code in your A06 sub-tab so that harvest report that is generated uses the Start and End dates, and the Field drop-down.
40. Commit your changes to your feature branch with a meaningful commit message and push it to your origin.

