07 - FarmOS API Technology Spike

COMP290 – Large Scale and Open Source Software Development
Dickinson College

Name:

Introduction:

At the end of activity A06 you were able to use Axios and the FarmOSAPI library functions to retrieve 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. In the process you'll learn a little more about Vue and you'll fill out more of the Sample Harvest Report by adding real harvest data.

The farmOS Data Architecture and Interface:

The Architecture page (https://v1.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.

1. 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.

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_aI

3. 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?

4. 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).

5. Use the menus in FarmData2 to answer the following questions.

a. What four types of assets exist in FarmData2?

b. What 13 types of logs exist in FarmData2?

c. Who are the 10 people (i.e. accounts) that exist the Developer Install of FarmData2?

d. What are the three different roles that people can have in FarmData2?

farmOS API Endpoints:

As you saw in activity #6, farmOS provides an API for accessing its data. The *FarmOS API Reference* page provides documentation for this API:

- https://v1.farmos.org/development/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 worry about 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 will not be particularly relevant to us. Instead, you will continue using Hoppscotch to explore and experiment with the farmOS API.

6. 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 user with Farm Worker and/or Farm Manager roles.
- 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.

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

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

a. Based on what you have seen in the FarmData2 menu options, what are some additional endpoints that you expect would exist for Assets?

b. Similarly, what are some additional endpoints that you expect 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.

9. 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?

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

a. Using the farmOS API Reference, give the endpoint that we would use to access harvest data through the API.

b. Use Hoppscotch to make a request from the URL you found in part a. The "Response Body" that is returned should be an object with the first few properties being `self`, `first`, `last`... If not, revisit part a and check the syntax of your request in Hoppscotch.

c. When the response to an API request will contain a large amount of data it will be split into *pages*. The response to the initial API request then returns the first page of results and

information about how to find the additional pages. How many pages of results are there for the request for harvest logs that you made?

d. What endpoint would you use for an API request to retrieve the next page of results?

11. The `list` property in the "Response Body" contains all of the harvest information. The value of `list` is an array of objects, where each object represents one harvest log.

a. How can you tell that the value of `list` is an array? Hint: Remember that JSON uses the same syntax for arrays and objects as JavaScript.

b. How can you tell that the elements contained in the `list` array are objects?

12. Looking at the "Response Body" you can see that each harvest log in the `list` array contains properties such as `inventory`, `movement`, `quantity`, etc… some of which you'll notice are also objects or arrays of objects. In order to use the values in the harvest logs we'll need to be able to reference them using dot and array index notation.

For example, `list[0].id` would reference the `id` property of the 0th harvest log object (which has value 1266). Use the "Response Body" and array and dot notation to answer the following questions:

a. What value is referred to by `list[0].timestamp`?

b. What value is referred to by `list[0].type`?

c. What value is referred to by `list[0].uid.id`?

d. What value is referred to by `list[0].quantity[2].label`?

e. Give a reference to the `id` property of the 5th harvest log.

f. Give a reference to the `id` property of the `uid` object of the 0th harvest log. Hint: The value of this `id` is 6.

g. Give a reference to the `name` property of the object in the `area` array of the 0th harvest log. Hint: The name there is "GHANA-2".

Dates and Times in FarmData2:

A lot of the reporting and logging features of FarmData2 will involve the use of dates. For example, in your Sample Harvest Report spike the user is able to specify the start and end date for the report. Similarly, in every log in FarmData2 will have a date associated with it indicating when the corresponding event (seeding, transplanting, harvesting) occurred. Thus, in order to work with these logs we will need to understand how FarmData2 handles dates.

13. Dates in FarmData2 are stored using *timestamps*. A timestamp is an integer that represents a date using *Unix Epoch Time*. Using your favorite search engine, read a little about Unix Epoch

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

14. 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://v1.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?

15. Use the time converter at: https://www.unixtimestamp.com/ to answer the following questions:

a. What local (EDT) date and time does the timestamp from the 0th harvest log you found earlier represent? Hint: Use your answer to #12a.

b. What timestamp would appear in a log that was created May 5, 2020 at 00:00:00 EDT?

c. What timestamp would appear in a log that was created May 15, 2020 at 00:00:00 EDT?

You may have noticed that all of the timestamps in question #15 have the time of 00:00:00. This is the first second of the day (i.e. just after midnight the night before). FarmData2 uses this time because it tracks only the date on which events occurred, not the specific time of day. This also implies that all events that occur on the same day will have the same timestamp, regardless of what time they occurred.

Using Timestamps in FarmOS API Requests:

Now that we know how FarmData2 represents dates as timestamps we can use them to narrow down our API requests so that we get just the logs that we want for our report. We do this by appending *query parameters* to the API endpoint. A query parameter is a string like `name=value` where `name` and `value` are meaningful to the API. For example, the farmOS API endpoint `/log.json?type=farm_harvest`, that we used earlier, has the query parameter `type=farm_harvest`. This tells the farmOS API that we only want logs where the `type` property is equal to `farm_harvest` (see #12b).

We can send multiple query parameters to an endpoint by separating them with an `&` character. For example, the following will request all of the logs with the `type=farm_havest` and `timestamp=1557201600` (i.e. May 7th 2019, 00:00:00 EDT):

```
/log.json?type=farm_harvest&timestamp=1557201600
```

16. Give a request including the endpoint and query parameters that will request all harvest logs from May 5, 2020.

17. Use Hoppscotch to test your request from #16. Don't forget you'll need to add `http://localhost` before the request. How many harvest logs are there for May 5, 2020? Hint: You can use the little triangles in the "Response Body" to collapse the harvest log objects to make them easier to count how many harvest log objects are in the `list` array:

Dealing with Date Ranges:

The above examples let us use query parameters to get logs from a specific day using its timestamp. The farmOS API provides an extended query parameter syntax that will allow us to specify a range of dates. For example, the following request asks for all harvest logs that occurred on or before May 7th:

```
/log.json?type=farm_harvest&timestamp[le]=1557201600
```

The `[le]` included in the query parameter indicates that we want the logs of all timestamps less than or equal to the provided value (i.e. occurring on May 7th 2019 or before). The farmOS API allows us to use any of the following with our query parameters:

- `[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

18. Give a request that would retrieve all of the harvest logs that occur between May 5th 2020 and May 15th 2020, inclusive of those days. Hint: Use `&` to add another query parameter to the request.

19. Use Hoppscotch to test your request from #18. Don't forget you'll need to add `http://localhost` before the endpoint and query parameters. What are the `ids` of the first and last harvest logs in the `list` property of the "Response Body"?

Adding Another Sub-Tab:

The sequence of spikes that follow will guide you through the process of making the Harvest Report table display live data from the FarmData2 database.

20. 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.

21. Make sure you have your feature branch checked out. Add another new sub-tab named API2 to the FD2 Example tab. Have the contents of this new tab be provided by the file api2.html. Make a copy of your api1.html file into the api2.html file. Don't forget to clear the Drupal cache when you are done. The result should be that you now have an API2 sub-tab that

is (for now) identical to your API1 sub-tab. You'll be working on the API2 tab throughout this activity.

22. 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.

Adding to the Harvest Report - Spike 1:

In order to add live data to the Harvest Report table we will need to make request to the farmOS API for all of the harvest logs that fall between the dates set by the user in the UI. The next few activities break that down into smaller steps and verify the results of each one along the way.

23. Recall that when requesting harvest logs from the farmOS API we used the timestamps to specify dates (e.g. #16, #18). However, what we have in our Vue instance are dates in the "YYYY-MM-DD" format that get their values from the start and end date UI elements (via v-model bindings). So, we need to convert the "YYYY-MM-DD" dates to their corresponding timestamps. Working with dates and timestamps is a common operation, so FarmData2 uses the popular dayjs library (https://day.js.org/).

The following JavaScript statement uses the dayjs library to convert a date from "YYYY-MM-DD" stored in the variable `myDate` to a timestamp:

```
let timestamp = dayjs(myDate).unix()
```

Add statements to the click handler for the "Generate Report" button that convert the start and end dates to timestamps. Display these time stamps using `console.log` statements so that you can test your code using some of the dates from earlier (see #15).

24. Next, add code to the click handler that builds a string holding the request (i.e. the endpoint and query parameters) that you'll need to use to request the harvest logs. Use a `console.log` statement to display the request. When this is working, the request should look something like the following:

```
/log.json?type=farm_harvest&timestamp[ge]=1588651200&timestamp[le]=1589515200
```

When you are testing, be sure that the timestamps in your request agree with those in the start and end date UI elements.

25. Now we need to make the API request to the farmOS API. You may recall that when making request it is possible that large responses will be split into multiple pages. Gathering multiple pages of responses can be tedious, so the FarmOSAPI library provides the `getAllPages` convenience function that does this for us.

Open the documentation for the FarmOSAPI in your browser and find the first example for the `getAllPages` function in its detailed documentation:

- `farmdata2_modules/fd2_tabs/resources/doc/index.html`

a. The first example for `getAllPages` requires an existing array to collect the resulting logs. Create new property in your Vue data for these results and initialize it to an empty array.

b. Using the example code in the documentation for `getAllPages` as a guide, add code to the click handler for the "Generate Report" button that request the harvest logs

c. Now let's check the response. Reload the page and use the Vue DevTools to inspect the array that you added to the `data` in part a. Before clicking the "Generate Report" button it should be empty. Set the start and end dates using those from question #18 and click the "Generate Report" button. Check that the array is populated with the appropriate harvest logs by checking the ids of the first and last one. Because you used the dates from question #18, these id's should agree with the ones you reported in question #19.

26. 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.

Adding to the Harvest Report - Spike 2:

27. We now have the data that is needed to populate the Harvest Report table. However, it needs to be transformed into the format expected by the v-for that you used to create the table. That is, it needs to be an array of objects with the properties date, area, crop, yield and units, as we defined it back in Activity #4. Since we are transforming data that we already have into a different format just for display purposes, we'll use a computed property. This computed property will process the harvest logs and return an array containing just the data that we want to appear in the table.

a. Add the following computed property to your Vue instance. Change the name in the yellow highlight so that the `for` loop will use the array of harvest logs in your Vue data. Note that the `for…of` loop in JavaScript is like a `for…each` loop in Java or a `for…in` loop in Python. In the code below the body of the loop is executed once with the variable `log` equal to each element of the array `this.harvestLogs`. Thus, each iteration of the loop adds one object to the `tableRows` array.

```
harvestReportRows() {
   let tableRows = []
   for(let log of this.harvestLogs) {
      let tableRow = {
         date: log.timestamp,
      }
```

```
        tableRows.push(tableRow)
      }
      return tableRows
    }
```

b. Update the Vue directives (e.g. `v-if` and `v-for`) in the HTML that generates your Harvest Report table so that it uses the data in the `harvestReportRows` computed property to generate the table.

c. Reload the page and generate a Harvest Report. You should notice two things. First, most of the cells in the table are now empty. Second, the Date column now contains timestamps, which are not very human friendly. You'll fix the first issue in this exercise and (optionally) the second in the next.

Why are the Area, Crop, Yield and Units columns empty?

d. Add code to the `harvestReportRows` computed property so that the Area column is filled in. To do this you'll need to add an `area` property to the `tableRow` object in the `harvestReportRows` computed function. Then use array and dot notation to access the property that contains the name of the area. Hint: Use Hoppscotch to fetch a harvest log that you can look at it to find the property for the area name.

e. Add code to the `harvestReportRows` computed property so that the Yield and Units column are filled in. Hint: Each harvest log contains an array of `quantity` objects. One of those objects has the `label` harvest. That object has the values you want here.

f. Add code to the `harvestReportRows` computed property so that the Crop column is filled in. Due to some peculiarities of the farmOS API, accessing the name of the crop that was harvested requires some extra work. If you look through a harvest log, you might notice that the crop name appears as part of the `name` property, but that there is no property that gives just the name of the crop. It might be tempting to just extract the crop name from the `name` property, there are a number of reasons for not doing that. The right way to get the name of the crop is to use the `data.crop_tid` property and the Map from the crop id to the crop name. Recall that you saved this map in a data property of your Vue instance in the last activity. Add a statement similar to the one below to add the crop name. You'll need to replace the yellow highlighted part with the name of the data property holding the Map in your Vue instance.

```
crop = this.idToCropMap.get(log.data.crop_tid)
```

g. Optional: The Harvest Report table should now be filled in, but the date is still being shown as a timestamp instead of a nice human readable date. The dayjs library that we used to convert the dates to timestamps earlier also provides a way to convert timestamps back to dates. Do some research on the dayjs library to figure out how to convert a timestamp into a formatted date. Then add code to the `harvestReportRows` computed property so that the date appears in a human readable format.

28. 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.

Adding to the Harvest Report - Spike 3:

Optional: All of the challenges in this spike are optional.

29. The work you've done so far gets you most of the way to a simplified Harvest report. It also introduced all of the essential skills and concepts needed to begin basic FarmData2 Development. However, it still has a few quirks, and it is not quite complete. If you are interested in a few extra challenges, you can try the following exercises. You can do none, do one, do them all, do them in any order, whatever you like. Each one enhances the Harvest Report in some way, making it closer to fully functional. Because each challenge is a separate nameable unit of work you should make a separate commit with a meaningful commit message for each one that you complete.

a. Currently if you click the "Generate Report" button multiple times the set of harvest logs for each request is appended to the end of the table. Change the behavior of the "Generate Report" button so that the table displays only the logs for the most recent request. Hint: Just clear the harvest logs from the Vue data, let the computed property and data binding do the rest!

b. Add "All" options to the top of the Crop and Area dropdowns and make this the default option. Hint: Add code to the computed properties that are used to generate the options for these dropdowns.

c. Filter the results that appear in the Harvest Report table using the area that is selected in the Area dropdown. Add code to the `harvestReportRows` computed property that only adds rows for the harvest logs with the matching area.

d. Filter the results that appear in the Harvest Report table using the crop that is selected in the Crop dropdown. Add code to the `harvestReportRows` computed property that only adds rows for the harvest logs with the matching crop.

e. If you try to generate a report for a date range during which no harvests occurred (e.g. 03/01/2019 through 03/15/2019) then the Harvest Report appears but there is no table. It would be better to indicate to the user that there were no harvests during the selected

period. Add a message that appears in place of the table if there are no harvest logs for the selected period. Hint: Use conditional rendering.

f. It is possible to enter incomplete dates into the start or end date fields (e.g. mm/05/2019 or mm/dd/yyyy). Make it so that the "Generate Report" button is disabled if either of the data fields do not contain a valid date. Hint: Try binding the `disabled` attribute of the button to a new computed property that returns true when one of the dates in the Vue data is invalid.

g. In question #27.e you used an array index to get the "harvest" `quantity` object that contained the quantity and units that were harvested (e.g. 10 BUNCHES). Using a fixed index works but may be brittle in that at some point in the future the order of the `quantity` objects might change. If that were to happen then the index that you used would give the incorrect quantity object. To make the code less brittle, the FarmOSAPI contains a helper method that will get a quantity object based on its label (e.g. "harvest"). Use the documentation for the FarmOSAPI to find and learn about this function. Then incorporate it into your `harvestReportRows` computed property.

30. Optional: Push your feature branch containing your commits to your origin. Recall that this also updates your Draft Pull Request.

Optional: To help us improve and scope these activities for future semesters please consider providing the following feedback.

a. Approximately how much time did you spend on this activity outside of class time?

b. Please comment on any particular challenges you faced in completing this activity.