06 - FarmOS API Technology Spike

COMP290 – Large Scale and Open Source Software Development
Dickinson College

Name:

Introduction:

At the end of activity 05 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 how farmOS and FarmData2 organize data and how to use the farmOS/FarmData2 API via the FarmOSAPI library to access that data. You won't learn everything but hopefully you'll learn 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 your Harvest Report Spike by adding real harvest data from the database.

The farmOS / FarmData2 Data Architecture and the User Interface:

There is a strong connection between the user interface features provided by farmOS and the way that farmOS and FarmData2 represent the data that they use. Being able to see this connection will be helpful in finding APi endpoints that provide the corresponding data. These questions explore that relationship.

The farmOS Architecture page (https://v1.farmos.org/development/architecture/) provides a high-level overview of how farmOS and FarmData2 interact with Drupal to organize data. In particular, this page describes the Drupal *entity types* that farmOS uses. 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. But more importantly for us, these entities represent the main types of data that are used by farmOS / FarmData2.

Read the farmOS Architecture page and answer the following questions:

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

2. Which three of those entity types appear directly as items in the menu at the top of the FarmData2 interface? Note: If the menu items are not visible use the “hamburger menu” (☰) in the upper right corner to find the relevant menu items.

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 (14:45)

3. Use the “hamburger menu” (☰) and its sub-menus in farmOS/FarmData2 to answer the following questions about the data organization in farmOS/FarmData2.

    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 in 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 05, farmOS provides an API for accessing its data. We used the `/farm.json` endpoint from that API to get some basic information about the farm. The *FarmOS API Reference* page provides documentation for all of the endpoints that are provided by this API:

- https://v1.farmos.org/development/api/

This page begins with information on authentication. Because the FarmData2 features 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. The next section discusses the `/farm.json` endpoint that we have already used.

The *API Version* section of the page is what will be of interest to us here. The early examples that use the `curl` command show how to make farmOS API requests using the command line tool `curl`. While the `curl` tool is useful in some contexts, they are using it in the same way that we have been using Hoppscotch to explore and experiment with the farmOS API. We will continue to use Hoppscotch so you can skip over this section.

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 farmOS/FarmData2 interface.

The questions in this section will have you explore the farmOS/FarmData2 interface and connect what you see there to the API endpoints that exist.

4. 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 window and load the Hoppscotch site.
     - https://hoppscotch.io

5. Compare the endpoints listed for Assets and Logs to the "Hamburger Menu" options in FarmData2. What do you notice?

6. 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 might guess would exist for Assets?

b. Similarly, what are some additional endpoints that you might guess 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.

7. For example, imagine we want to find all harvests from a particular field. Use the farmOS/FarmData2 interface to find all harvest logs from the field Jasmine-1. Hint: Go to the harvest logs and use the "Filters" tab.

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?

8. 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 log 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 contains a large amount of data the server will split that response 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?

9. 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?

10. 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 properties and 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 `log_owner` object of the 0th harvest log. Hint: The value of this `id` is 3.

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”. Hint2: the `area` property is an array.

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, 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 farmOS/FarmData2 handles dates.

11. Dates in farmOS/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.

12. 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 the set of "standard fields" that appear in most log types under the *Creating Logs* heading.

Which of these three timestamps represents the time at which the harvest occurred?

13. 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 #10a.

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 #13 have the time of 00:00:00 (the first second of the day). All of the dates in farmOS/FarmData2 will have this property because it tracks only the date on which events occurred, not the specific time of day. This also means that all events that occur on the same day will have the same timestamp, regardless of what time in the day that they occurred.

Using Timestamps in FarmOS API Requests:

Now that we know how farmOS/FarmData2 represents dates as timestamps we can use them in our API requests to help specify 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 you used earlier, had the query parameter `type=farm_harvest`. This query parameter told the farmOS API that we only wanted logs where the `type` property is equal to `farm_harvest`.

Many endpoints will allow us to include multiple query parameters as long as we separate them with an `&` character.

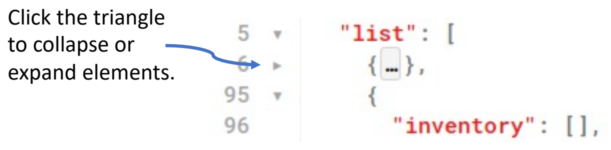
For example, the following query parameters will request all of the logs where the `type=farm_havest` and `timestamp=1557201600` (i.e. May 7th 2019, 00:00:00 EDT):

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

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

15. Use Hoppscotch to test your request from #14. Don't forget that you will need to include the prefix `http://fd2_farmadat2` before the endpoint and query parameters so that the request is sent to the correct server.

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:



```
"list": [
  {…},
  {
    "inventory": [],
```

<u>Dealing with Date Ranges:</u>

The above examples let us use query parameters to get logs from a specific day using its timestamp. The farmOS API also 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 all of the logs with timestamps that are less than or equal (`le`) 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

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

17. Use Hoppscotch to test your request from #16. Don't forget you'll need to add `http://fd2_farmdata2` 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"?

<u>Synchronizing with the Upstream:</u>

18. Some time has passed since you created your fork and clone of the upstream FD2School-FarmData2 repository. It is possible that there have been updates to the upstream since you did so. So, as when working on any fork and clone it is important to synchronize your main branch with the upstream so that you have all of the recent changes.

a. Synchronize your local and origin FarmData2 repositories and your feature branch with the upstream. The steps you will need to do this include are:

1. Switch to the `main` branch.
2. Pull the `main` branch from the upstream.
3. Push the `main` branch to your origin.

If you don't remember the commands for this, you can refer back to the previous activity where you will have listed them.

b. Now because your `main` branch has been updated, you will need to merge those changes into the feature branch that you created in the prior activity. To merge the `main` branch into your feature branch:

1. Switch to your feature branch from the prior assignment (`<name>-05-API`).
2. Merge the `main` branch into your feature branch.
3. Resolve any conflicts that arise (If you are just doing the FD2School activities there should not be any).

If you don't remember the commands for this, you can refer back to the previous activity where you will have listed them.

c. Your work on this assignment builds from what you did in the prior assignment. So you now need to:

1. Switch to your prior feature branch `<name>-05-API` (if you are not there already).
2. Create a new feature branch named `<name>-06-API2` from your prior feature branch.
3. Switch to your new feature branch.

You will do all of your work for this activity in this feature branch.

If you don't remember the commands for this, you can refer back to the previous activity where you will have listed them.

<u>Adding a New FarmData2 School Sub-tab:</u>

19. Make sure you have your feature branch checked out and then add another new sub-tab named **API2** to the FD2 School tab. Have the contents of this new tab be contained in the directory `api` with the page content provided by the file `api2.html`. Make a copy of your `api.html` file into a file named `api2/api2.html`. Don't forget to clear the Drupal cache when you are done. The result should be that you now have four sub-tabs in the FD2 School tab: HTML, Vue1, Vue2, API and API2. For now, your API and API2 tabs will be exactly the same. You'll be modifying and extending the API2 tab throughout this activity.

20. Commit your changes to your feature branch with a meaningful commit message that describes what you have done and push it to your origin.

21. On GitHub create a *Draft Pull Request* for your new feature branch to the upstream FD2School-FarmData2 repository. Be sure to link your PR to the issue for this activity by including a line like the following in the body of your PR:

    Addresses #??

Replace the ?? by the issue number in the FD2School-FarmData2 issue tracker for this assignment.

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. This style of work where you verify each step along the way before going on to the next one is often called *incremental development*. Incremental development can significantly reduce debugging effort because if something doesn't work, then you know that it very likely is the most recent thing that you added.

22. Recall that when requesting harvest logs from the farmOS API we used the timestamps to specify dates (e.g. #14-#17). However, the start and end date HTML elements give us their dates in the form "YYYY-MM-DD". So, we need to convert the "YYYY-MM-DD" dates to their corresponding timestamps before we can use them in an API request.

Working with dates and timestamps is a common operation, so FarmData2 uses the popular library *dayjs* (https://day.js.org/).

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

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

Add statements to the `@click` handler for the "Generate Report" button that do the following:

- Convert the start and end dates from your Vue data instance to timestamps saved in local variables.
- Display the converted time stamps using `console.log` statements.
- Test your conversions using the dates you converted earlier (E.g. May 5, 2020 and May 15, 2020) as the start and end dates by looking at the output in the DevTools console.

23. 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.
- Display that string using `console.log`.
- Check that the request you created is correct. The output in the console should look something like the following depending on the dates you enter:

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

You should double check that the timestamps in your request agree with those you entered in the start and end date HTML elements.

24. Now we need to make the API request to the farmOS API. We could use Axios directly to make this request (as we did with `/farm.json`). However, like with the crop and area maps in the previous activity, the FarmOSAPI library provides a convenience function that we can use.

Open the documentation for the FarmOSAPI in your browser (if you don't remember how see farmdata2/README.md). Find the documentation for the `getAllPages` function and read the first sentence of its description.

What does the `getAllPages` function do?

25. As we saw earlier, when we make request that returns a lot of data that data will be split into multiple pages. The server returns the first page of the response and then it is our responsibility to request the next page and the next page, etc until we get all of the data. That process of gathering these multiple pages of responses is tedious. So, the `getAllPages` convenience function was created to do this for us. We give it an endpoint and it does the work of making the requests for all of the pages.

a. Scroll down to the first code example for `getAllPages`. This example shows how `getAllPages` can be used to fill an existing array with all pages of the result. Create new property in your Vue data that will hold these results and initialize it to an empty array.

b. Adapt the first code example in the documentation for `getAllPages` to add code to the `@click` handler for the "Generate Report" button to request the harvest logs between the start and end dates for the report and store them into the array you created in part a. Be sure to print an error message to the console if an error occurs during the request.

c. Now let's check that it works. 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. Use the default start and end dates and click the "Generate Report" button. Check that the array is populated with the appropriate harvest logs. These should be the same ones you saw in Hoppscotch in question #17 earlier, so you can check the first and last id's to be sure.

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. You now have the data that is needed to populate the Harvest Report table stored in the `data` of your `Vue` instance. However, it needs to be transformed into the format expected by the `v-for` that you use to create the rows in the report table. That is, we need to use the full harvest logs that the API gave us to create the array of objects with the properties `date`, `area`, `crop`, `yield` and `units`.

To do this you will transform the data that we already have (the full harvest logs) into a different format (the array with `date`, `area`, etc) using a Vue computed property. Because the new array is just for display, a computed property is the right choice for this task.

a. As a start adapt the following computed property and add it to your Vue instance. You'll need to at least change the name that is highlighted in yellow so that the `for` loop will use the array of harvest logs that you retrieved from the API and stored 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 will add 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. Once you have added the `harvestReportRows` computed property, reload the page and use the Vue DevTools to check that it is working. You should see the `harvestReportRows` computed property and it should have one array entry for every harvest log and each entry should contain the timestamp.

c. Now update the Vue directives (e.g. `v-if` and `v-for`) in the HTML that generates your Harvest Report table so that the data in the `harvestReportRows` computed property appears in the table.

d. 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.

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

e. Add code to the `harvestReportRows` computed property so that it adds the `area` property to the `tableRow` object. Hint: Use Hoppscotch to fetch a harvest log and use that to figure out the dot notation you need to access the property that contains the name of the area. Hint: Notice that the `area` property of a harvest log is an array. Reload your page and confirm that the Area column is now filled in.

f. Add code to the `harvestReportRows` computed property so that the Yield and Units columns 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. Reload your page and confirm that the Yield and Units columns are now filled in.

28. The Crop is the only column remaining to be filled in the harvest report table. We are doing it last because it is a little more complicated.

If you look through a harvest log, you will 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. So it might be tempting to use a substring operation to extract the crop name from the `name` property and use that in the table. But there are a number of reasons for not doing that. The main one is that because logs can be edited after creation, the crop name in the log may not match the actual crop that was planted.

The right way to get the name of the crop is to use the `data.crop_tid` property that appears in the log. Then use the `Map` from the crop id to the crop name to convert the `crop_tid` into the name of the crop. You may recall that in the last activity you used the `getIDToCropMap` function in the FarmOSAPI library to request this map and then saved it in a data property of your Vue instance in the last activity.

Adapt the statement below to use the `Map` add the crop name to your data row. 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)
```

Reload your page and confirm that the Yield and Units columns are now filled in.

29. All of the columns in your Harvest Report table should now be filled in. But notice that 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.

The dayjs library that we used to convert the date to a timestamp can also help us convert timestamps to dates. The following pages in the dayjs documentation will help:

- Parse -> Unix Timestamp (seconds):
  - https://day.js.org/docs/en/parse/unix-timestamp
- Display -> Format:
  - https://day.js.org/docs/en/display/format

Use the documentation in the above pages and modify your code in the `harvestReportRows` computed property so that the date appears in a human readable format that is consistent with the start and end date HTML elements.

Reload your page and confirm that the Date column now displays a human readable date.

30. You should now also be sure to remove the code from the earlier activities that added "fake" harvest logs to you report and any `console.log` statements that you included for testing since they are no longer needed.

31. Commit the changes you have made to fill in the harvest report table 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:

32. Modify the `harvestReportRows` computed property so that only rows matching the crop in the crop dropdown are displayed in the table. Hint: Check the name of the crop in each row against the crop in the dropdown and only add the row if they match.

Reload your page and confirm that the harvest report table is now filtered to show only the selected crop.

33. Commit the changes you have made to fill in the harvest report table to your feature branch with a meaningful commit message and push it to your origin. Recall that this also updates your Draft Pull Request.

Optional Challenges - Spike 4:

The work you've done so far gets you most of the way to a simplified Harvest report. 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. All of the challenges in this spike are optional. 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 meaningful way, making it closer to fully functional.

If you choose to do these challenges, you should make a separate commit with a meaningful commit message for each challenge that you complete. Also, be sure to update your PR by pushing your feature branch to your origin when you are done.

34. Currently if you click the "Generate Report" button multiple times the set of harvest logs that appear in the report table will be duplicated for each click. This is because each click generates a new API request via the `getAllPages` function. This function appends the results of that request to the array in your Vue `data`. The Vue computed property then filters that array and generates the rows for the table again, which now includes the duplicated rows. 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` before making the new request and let the computed property and data binding do the rest!

35. Add "All" options to the top of the Crop and Area dropdowns and make this the default option. When all is "All" is selected the report table should display rows for all crops. Hint: Add code to the computed properties that are used to generate the options for these dropdowns.

36. Filter the results that appear in the Harvest Report table using the area that is selected in the Area dropdown in addition to the Crop dropdown. Hint: Add code to the `harvestReportRows` computed property that only adds rows for the harvest logs where both the crop and the area match the dropdowns.

37. 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.

38. You might notice that once you have generated a report, changing the start and end date do not have any effect on the table that is displayed unless you click the "Generate Report" button again. This could potentially be confusing to a user. One solution would be to clear the report anytime a date is changed. Use an `@click` event handler to clear and hide the report when the user clicks on the start or end date inputs. Clicking the "Generate Report" button will then generate a new report with the new dates.

39. In question #27.f 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.

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.