

05 - Vue Events and JavaScript Functions Technology Spike

COMP290 – Large Scale and Open Source Software Development
Dickinson College

Name:

Introduction:

In the last activity you learned about *Vue Data Binding* and how Vue + JavaScript can be used to build pages with content that is rendered from the *Vue instance* (a JavaScript object). You saw that when changes are made to the Vue instance through the DevTools console or the Vue DevTools, the rendered page also changes. In this set of activities, you will build upon that by having JavaScript code modify the Vue instance in response to events (e.g. button clicks). You'll also see another way to do data binding, a few new Vue directives, and you'll learn some more JavaScript. Then in the next two activities you'll learn how to use JavaScript to get data from web services using APIs which will allow you to bring live data from the FarmData2 database into your harvest report.

Vue School:

We'll be continuing with the free *Vue.js Fundamentals* course (<https://vueschool.io/courses/vuejs-fundamentals>) from Vue School (<https://vueschool.io/>). It is not required, but if you would also like a textual source that covers much of the same material you might find the *Introduction to the Vue.js Guide* (<https://vuejs.org/v2/guide/index.html>) helpful.

Handling User Events:

Find the *User Events (4:04)* video in the free *Vue.js Fundamentals* course. Watch that video and complete the activities below by continuing your `vuespike.html` file from A04.

1. What Vue directive do you add to a button in order to respond to a click? What is the shorthand for this directive?

2. Add HTML to your `vuespike.html` that adds a new label, field and button as shown below:



This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-nc-sa/4.0/).

Name:

Give the lines of HTML that you added to produce this label, field and button.

3. Now, modify your page so that when the button is clicked whatever is in the text field is added to the list of names. As shown in the video you will need to add a property to your Vue instance, bind it to the text field and add an event handler to the button.

a. Give the updated HTML for your text field that includes the data binding.

b. Give the code that defines and initializes the new property that you added to your Vue instance.

c. Give the updated HTML element for your button that includes the call to the event handler.

4. Modify your text field it so that the name is also added if you press Enter or Return. Give the HTML element for your text field with this update.

5. Hopefully that all works now. But from a User Interface/User Experience (UI/UX) perspective it has some issues. We'll see them here and then fix them shortly.

a. Enter a name in the text field and then click the "Add Name" button or press Enter/Return multiple times. What happens?



b. What happens if the text field is empty when you press Enter/Return or click Add name?

6. Let's also gather a little more information about the behavior of your text field, Vue instance property and button that will help when improving the UI/UX later. You'll want to open and use the Vue DevTools for these. If you don't recall how to open the Vue DevTools, check Activity 04 or the User Inputs & Vue Devtools Video (<https://vueschool.io/lessons/vuejs-user-inputs-vue-devtools>).

a. What is the initial value that is stored in the Vue instance property that is bound to your text field?

b. Type something into the text field and then delete it. What value is in the bound property in Vue instance now?

c. If your answer to parts a and b are not the same, modify the value assigned to the property in your Vue instance so that they are. This will simplify things later. Give the line of code from your Vue instance that creates and initializes the property bound to your text field.

Adding to the Harvest Report - Spike 1:

7. 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). Give the sequence of commands that you used.



8. Make sure you have your feature branch checked out. Add another new sub-tab named Vue2 to the FD2 Example tab. Have the contents of this new tab be provided by the file `vue2.html`. Make a copy of your `vue1.html` file into the `vue2.html` file. Don't forget to clear the Drupal cache when you are done. The result should be that you now have an Vue2 sub-tab that is (for now) identical to your Vue1 sub-tab. You'll be working on the Vue2 tab throughout this activity.

9. 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.

10. In your `vue2.html` page in FarmData2:

- Set the property in your Vue instance that holds the harvest logs to be an empty array.
- Then have the handler called when the "Generate Report" button is clicked add an object representing harvest log shown below to the array in the Vue instance:

| Date | Area | Crop | Yield | Units |
|------------|---------|------|-------|---------|
| 05/01/2018 | Orion-3 | Kale | 12 | Bunches |

Note: You'll need to be careful with single and double quotes when writing the string for the click handler. I suggest double quotes for the string and single quotes for the object property values.

When this works correctly, the table in the harvest report should initially be empty (just the headers). Then when the "Generate Report" button is clicked the new row should appear in the table. If you click the button multiple times, then multiple copies of the row should appear. Not quite the real-deal yet, but we are getting closer. Eventually, the JavaScript that runs when this button is clicked will retrieve data from the FarmData2 database using a web API and load it into the Vue instance.

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

Vue Methods and JavaScript Functions:

Find the *Vue Methods (3:00)* video in the free *Vue.js Fundamentals* course. Watch that video and complete the activities below by continuing to modify your `vuespike.html` file.

12. Mimicking what is done in the video, move the code that runs when your "Add Name" button is clicked from the button tag to a function in the methods object in your Vue instance.

- a. Give the full methods object that you added to your Vue instance here including the function that handles the button click.



b. Give the HTML for your button tag after this change.

c. Give the HTML for your text field after this change.

13. Add code to your function from #12 so that the text field is cleared after the name is added to the list. Hint: Your function should not change the text field directly. It should modify the Vue instance and let the data binding do the work! Give the line you added to your function.

14. Extend your function from #13 so that a name is only added to the list if there is something in the text field (i.e. no more adding blanks). To complete this you'll need to use a JavaScript conditional (`if`) statement. Using what you know about other programming languages a quick skim of the MDN resource below should be enough to get you started with JavaScript conditionals:

- https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/conditionals

Give the conditional statement that you added to your function here. Hint: Your answer to question #9 will also be helpful here.

15. Optional: Add a text field and a button for adding a new card to the list. The input to the text field should contain information for a new card (e.g. A H or 3 D, etc.). Your JavaScript function should split the text and add a new object to the array of cards in your Vue instance. You might find the MDN page on Useful String Methods helpful:

- https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Useful_string_methods

Adding to the Harvest Report - Spike 2:



16. In your `vue2.html` file in `FarmData2`, move the JavaScript code that handles the “Generate Report” button click to a function in the methods object in your Vue instance. Hint: Don’t forget the `this`!

17. Optional: Modify the handler function so that it adds a different harvest record to the Vue instance (and thus to the table as well) each time the button is clicked. Your code should add at least three different records. Hint: Use a JavaScript conditional (`if/else`) and the length of the array to add a different record each time. A quick scan of the MDN page on arrays should reveal how to determine the length of an array:

- https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Arrays

Once your function has added all of the records, additional clicks of the button should not do anything.

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

v-for Indices and Functions with Parameters:

Sometimes when using the `v-for` directive to render a list of items it will be useful to have a variable telling us the index of each element as we are using it. The `v-for` directive has another form that will provide this index. This section will introduce you to this form of the `v-for` directive and show you a handy way to use it.

19. Modify the `li` tag that generates the list of names in your `vuespike.html` page so that it is similar to the one shown below. You may need to adapt it depending upon what you called the array of names in your Vue instance.

```
<li v-for='(name,index) in names'>{{ name + ' (' + index + ')' }}
```

This modified format for the `v-for` directive defines two variables, `name` and `index`, as it iterates over the array. The `name` variable takes on each value in the `names` array, just as it did before. The `index` variable is a counter that starts at 0 for the first name and increasing by one as the `v-for` goes through each additional name.

20. Think about what you would expect the list to look like when you reload the page based on the change you made in question #19. Then reload the page.

a. Paste a screen shot of just the list of names below.



b. Was the output what you expected? Briefly explain why it was or was not what you expected.

We can use the `index` variable created not just in the double mustache but anywhere within the tag where it is defined (e.g. between `` and `` in this case). That can be very useful for creating additional HTML elements that know the index that they are associated with. For example, the `v-on:click` (i.e. `@click`) event handler of the following button uses the `index` as an argument to the `splice` array function:

```
<button type='button' @click='names.splice(index,1)'>Delete</button>
```

To understand what this line does, check the MDN documentation for the JavaScript `Array.splice` method. Pay particular attention to the `start` and `deleteCount` parameters.

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

21. Optional: Add the button tag above between the `` and `` tags that generate the list of names in your `vuespike.html` page. Again, you may have to adapt the statement to match the name of the array that holds the names in your Vue instance. Briefly explain what clicking the button does and the role that the `index` variable plays.

As you see with the `splice` method JavaScript functions can have parameters. The above call to `splice` passes the value of the `index` variable and the value 1 as values for the `splice` function's parameters.

The functions that we write ourselves can also have parameters. For example, consider the following function that might appear in the `methods` object of the Vue instance:

```
deleteName: function(nameIndex) {  
  this.names.splice(nameIndex, 1);  
}
```

This function accepts one argument as the value for the `nameIndex` parameter. The value of that parameter is then used in the call to `splice`.



22. Optional: Add the `deleteName` function above to the `methods` object of the Vue instance in your `vuespike.html` page.

23. Optional: Modify the button tag from question #21 so that it now calls your `deleteName` function instead of using `splice` directly. Be sure the call to the `deleteName` function in your button tag passes the index of the name to be deleted. Test your delete button to be sure it still deletes the name next to it. Paste the code for your button element that calls `deleteName` here.





Adding to the Harvest Report - Spike 3:

24. Add a column on the left of the harvest report that numbers the logs that appear in the table so that the report table will look like the following:

| Row | Date | Area | Crop | Yield | Units |
|-----|------------|---------|------|-------|---------|
| 1 | 2018-05-01 | Orion-3 | Kale | 12 | Bunches |
| 2 | 2018-05-01 | Orion-3 | Kale | 12 | Bunches |
| 3 | 2018-05-01 | Orion-3 | Kale | 12 | Bunches |

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

26. Optional: Now, when working with reports (e.g. like the Harvest Report) the farmer may want to be able to delete or edit records. So, each row of a report in FarmData2 will have an “Edit” and a “Delete” button something like shown below:

| date | crop | area | Tray/Direct | Hours | Num Workers | Varieties | Comments | User | Edit | Delete |
|------------|-----------------|------|-----------------|-------|-------------|--------------------|----------|---------|---|---|
| 2020-05-05 | LETTUCE-MES MIX | K | Direct Seedings | 0.02 | 1 | LX-20-1-UT (SULU) | | worker4 |  |  |
| 2020-05-05 | SPINACH | K | Direct Seedings | 0.02 | 1 | SH-20-3-UT (SPACE) | | worker4 |  |  |

Add a column to the harvest report table in your `vue2.html` file in FarmData2. That column should contain a delete button as shown below. When the delete button in a row is clicked, that row should be removed from the table. Hint: Your code should remove the row from the array in the Vue instance and let the data binding to the rest!

| Row | Date | Area | Crop | Yield | Units | Delete |
|-----|------------|---------|------|-------|---------|--------|
| 1 | 2018-05-01 | Orion-3 | Kale | 12 | Bunches | Delete |
| 2 | 2018-05-01 | Orion-3 | Kale | 12 | Bunches | Delete |



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

Conditional Rendering:

Find the *Conditional Rendering (3:13)* video in the free *Vue.js Fundamentals* course. Watch that video and complete the activities below by continuing your `vuespike.html`.

28. Set the value of the array you use to hold the list of names in your Vue instance in your `vuespike.html` file to be an empty array (E.g. something like: `names: []`). Now what appears in the names list when you reload the page?

29. Instead of having the list not appear at all (as it should have in #26), it would be better to display a message that there are no names yet. Add an `li` to the list using a `v-if` directive so that that message “No names yet!” appears in the list if no names have been added. This message should only appear when there are no names in the list. Confirm that it is hidden when you add a name and that it reappears if you delete all of the names. Give the HTML for your `li` element using `v-if` that displays the message.

30. Mimicking what was done in the video, modify your `vuespike.html` so that there are two buttons “Show Cards” and “Hide Cards”. By default, the page should not show the list of cards but should show the “Show Cards” button. When show cards is clicked the list of cards and the “Hide Cards” button should be displayed and the “Show Cards” button should be hidden. Hint: Add a state variable to your Vue instance and use it in the `v-if`, `v-else` and `@click` directives. Hint2: Enclose your list in a `div` that has a `v-if` directive to make it easier to show and hide the whole list. Give the HTML for your “Show Cards” and “Hide Cards” buttons.

31. Optional: Add a “Clear Names” button that appears when there are names in the list and is hidden when the names list is empty. When clicked the list of names should be made empty.

Adding to the Harvest Report - Spike 4:



32. Modify your `vue2.html` file in FarmData2 so that the table headings for the Harvest Report only appear when there are harvest logs to be displayed. If there are no harvest logs to be displayed, then a message should appear indicating that there are no matching records.

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

34. Optional: Modify your `vue2.html` file in FarmData2 so that the whole section of the page showing the Harvest Report only appears after the Generate Report button is clicked. The report section of the page should remain visible even if all of the rows of the table are deleted (i.e. once shown, the report will remain visible.) Hint: Enclose your entire report section in a `div` and add a new variable to your Vue instance.

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

36. Optional: Modify your `vue2.html` file in FarmData2 so that the first click of the Generate Report button just displays the report but does not add any harvest logs (i.e. the report should be empty when first displayed). Each additional click should then add an additional harvest log.

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

Attribute Bindings:

Find the *Attribute Bindings (1:41)* in the free *Vue.js Fundamentals* course. Watch that video and complete the activities below by continuing your `vuespike.html`.

38. What Vue directive is used to bind an attribute of an HTML element to a data value in the Vue instance? What is the shorthand for this directive?

39. Modify the Add Name button so that it is disabled until something is typed in the text field. Give the HTML for your updated button tag here.

40. Optional: The Clear Names button you added earlier (if you did that optional part) appearing and disappearing when the list changes is a little distracting and not the best UI/UX



design. Instead, modify the Clear Names button so that it is disabled when the list is empty and enabled when there are names in the list.

Adding to the Harvest Report - Spike 5:

You may have noticed that when working with the start and end dates for the Harvest Report it is possible to set a start date that comes after the end date or vice versa. This of course would result in an empty report. So, it would be better UI/UX design if it were not possible to choose a start date that comes after the end date or an end date that comes after the start date. Because the `v-bind` (i.e. `:`) Vue directive can be used with any HTML element attribute, it can help us with this.

41. Modify your `vue2.html` page so that the start date cannot be set to any date after the end date and that the end date cannot be set to be set to any date before the start date. Hint: Bind the appropriate min and max properties of the date elements.

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

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.

