Vue Proof of Concept 1: JSON and Basic Syntax

Objectives:

- Import JSON data
- Create a Vue app
- Use `v-text`, `v-html`, `v-model`, `v-on`, and `v-bind`
- Review ternary conditionals, `forEach()`, and arrow functions

Explore Starter Files

1. Open the **js/data.json** file and examine its contents. This is a standard JSON file with a root property (albums) that stores an array of objects – each with three properties.

2. Open the **css/poc1.css** file and examine its contents. Rules are already targeting specific HTML tags and IDs, so you'll need to make sure you code the HTML perfectly so that the rules work.

Code the HTML

1. Create an HTML file named: **index.html**

2. In addition to the standard HEAD content, link to the starter CSS file.

3. Include your name in the document title along with any other information you deem relevant.

4. Link to the CDN version of Vue. Be sure to link to the current version 3 iteration and not the generic 'whatever the current version is'. We don't want the version of Vue changing on us!

5. Link to a yet-to-be-created JavaScript file named: **vue-poc1.js**

6. Be sure to link to the CSS and both JS files in the HEAD, and ensure the browser waits until all HTML is loaded before executing any JS code.

7. Code the BODY content as shown below. Be careful of the IDs!

```
<body>
  <div id="app">
    <h1>App Name</h1>
    <p id="total">Albums found: <span>999</span></p>
    <input type="checkbox" id="date-box">
    <label for="date-box">Show dates</label><br>
    <span id="number">#</span>
    <a target="_blank">Album link</a>
    <span id="date">Release date</span>
    <hr>
    <button>Previous</button>
    <button>Next</button>
  </div>
</body>
```

8. When viewed in the browser, your page should look like this (notice the label styling changes when the checkbox is checked):

Fetch Data

1. Create the **js/vue-poc1.js** file and fetch the data from the external JSON file. Note the `fetch()` path is relative to the HTML file, not the JS file. This is a review of WEB-215 material.

```
fetch('js/data.json')
  .then(response => handleErrors(response))
  .then(data => fetchSuccess(data))
  .catch(error => fetchError(error));
```

2. Create the `handleErrors` function. This is a review of WEB-215 material.

```
function handleErrors(response) {
  if (!response.ok) {
    throw (response.status + ': ' + response.statusText);
  }
  return response.json();
}
```

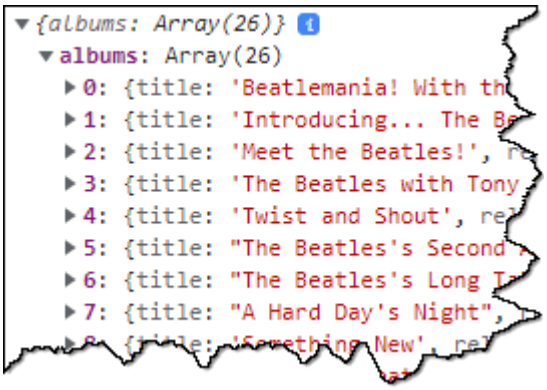
3. Create the `fetchError` function. For right now, it will just display the error message in the console. This is a review of WEB-215 material.

```
function fetchError(error) {
  console.log(error);
}
```

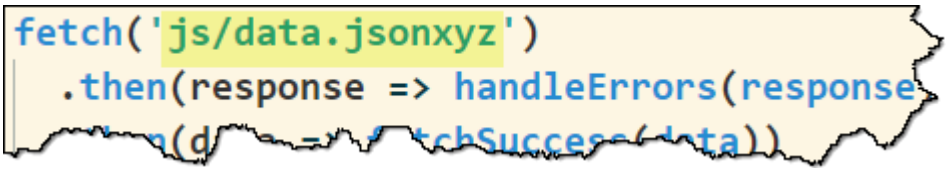
4. Create the `fetchSuccess` function. For right now, it will just display the data in the console. This is a review of WEB-215 material.

```
function fetchSuccess(dataFromServer) {
  console.log(dataFromServer);
}
```

5. Load the HTML file in your browser and examine the console. You should be getting data back from the `fetchSuccess` function.



6. Edit the `fetch()` statement to create an error situation:



7. Your console should not display the results from the `fetchError` function.

8. Leave the error alone for now so we can focus on the `fetchError` function output.

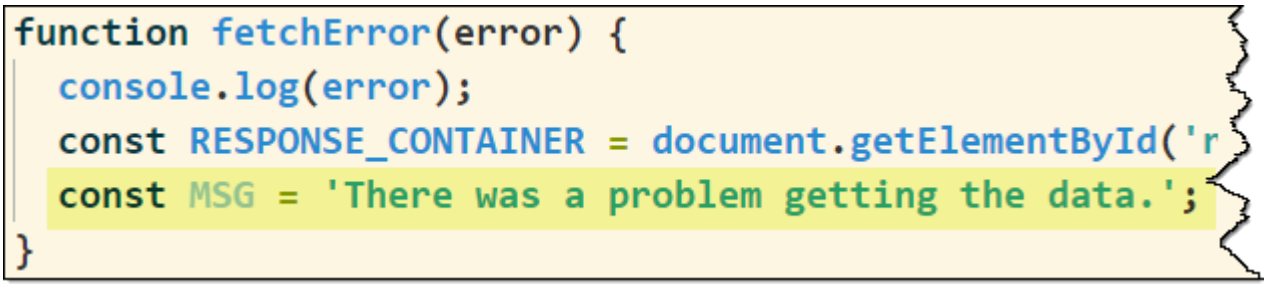
Error Rendering

Code everything in this section within the `fetchError` function body.

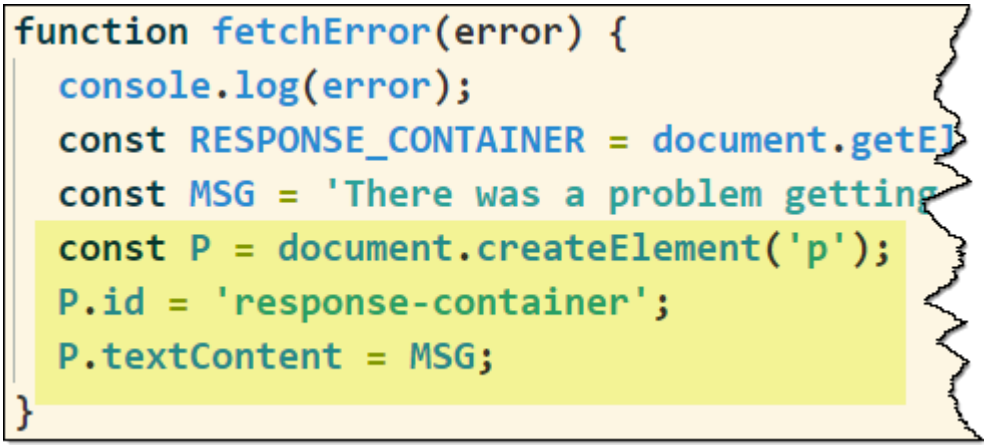
1. First, we'll plan for the future and delete any existing error message on the page. In a single statement:
    a. Create a variable to store a paragraph with an ID of response-container. Note this paragraph may or may not exist on the page. Use `getElementById` rather than `querySelector` because it's faster.
    b. If the paragraph exists, then remove it.
    c. If the paragraph doesn't exist, do nothing.

```
function fetchError(error) {
  console.log(error);
  const RESPONSE_CONTAINER = document.getElementById('response-container') ? RESPONSE_CONTAINER.remove() : '';
}
```

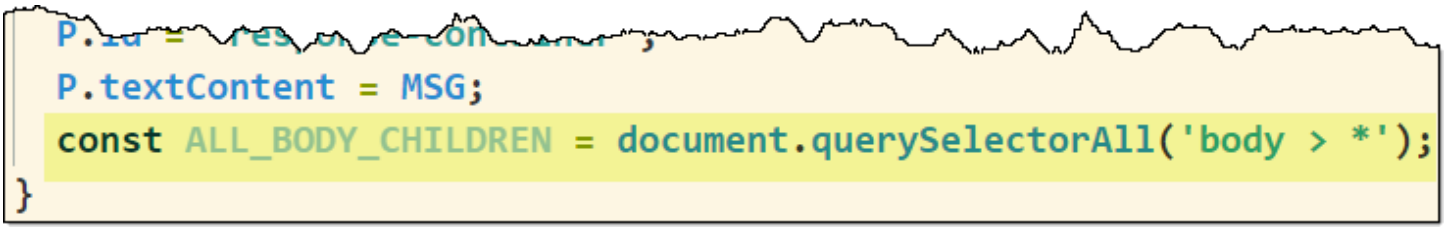
2. Create a new message to display as a user-friendly error message:



3. Create a new paragraph element, give it the ID specified earlier when we were checking to remove old errors, and load the message string into the paragraph.



4. Before displaying the error in the viewport, let's hide all existing content. We'll need to loop through every child of the <body>. We'll need to use `querySelectorAll` here because we need to use an advanced CSS selector



5. Loop through them all and hide them. Use `forEach()` to run the same function on all the elements. Remember that the first argument in `forEach()` is the current element. This is a good time to review arrow functions. (Alternatively, you could have used `el.remove()` to remove the elements from the DOM rather than just hiding them.)

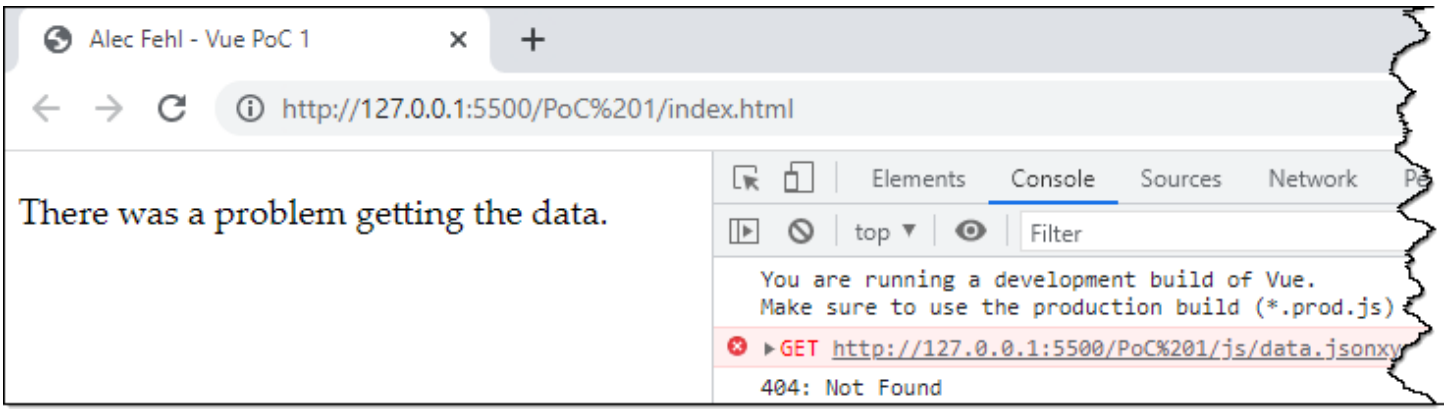
```
  P.textContent = MSG;
  const ALL_BODY_CHILDREN = document.querySelectorAll('body > *');
  ALL_BODY_CHILDREN.forEach(el => el.style.display = 'none');
}
```

6. Load the page in the browser. The viewport should be blank, but the console should still display the error message from the server.

7. Now that the body is empty, go ahead and display the user-friendly error message. Note we're using `getElementsByTagName` rather then `querySelector` because it's faster. Also – because `getElementsByTagName` returns a list of elements, we want to grab just the first one, thus the `[0]`.

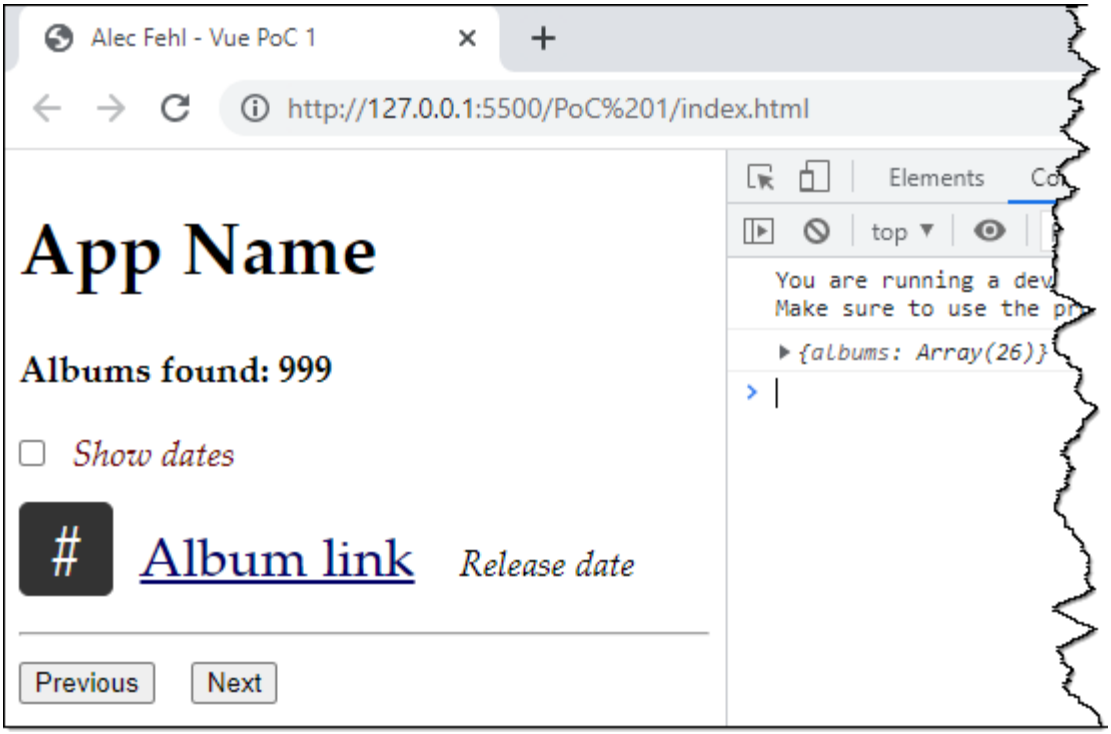
```
  const ALL_BODY_CHILDREN = document.querySelectorAll('body > *');
  ALL_BODY_CHILDREN.forEach(el => el.style.display = 'none');
  document.getElementsByTagName('body')[0].appendChild(P);
}
```

8. Test the page. Only the user-friendly message renders. The technical server message is still available to developers in the console.



9. Edit the opening `fetch()` statement to fix the error. The original rendering returns.

```
fetch('js/data.json')
  .then(response => handleErrors(response))
  .then(data => fetchSuccess(data
```



Break Time!

You may have realized – we haven't done anything with Vue yet other than linking to the library! This has all been a review of WEB-215. Go walk around the block and clear your head before moving on to the `fetchSuccess` function.

Success Rendering and Vue

1. Begin to use Vue by doing the following.
    a. Create a variable to store the Vue app
    b. Use the required Vue method, `data()`, to return data
    c. Return a single property (for starters)
    d. Keep track of all closing braces
    e. Mount the app in the HTML element with the ID of `#app` (the first DIV in your HTML)

```
function fetchSuccess(dataFromServer) {
  console.log(dataFromServer);
  const ALBUMS_APP = {
    data() {
      return {
        listName: 'The Beatles North American Albums'
      }; // end return
    } // end data()
  }; // end ALBUMS_APP
  Vue.createApp(ALBUMS_APP).mount('#app');
} // end fetchSuccess function
```

2. Edit the HTML H1 tag's content to replace the text *App Name* with data returned from Vue.

```
<body>
  <div id="app">
    <h1>{{ listName}}</h1>
    <p id="total">Albums found: <span>999</span></p>
    <input type="checkbox" id="date" ...
```

3. Test your page. You should get the following heading. If you don't, go back and find your mistake(s). This absolutely has to work before you move on!

Disclosure:

It's assumed you've been through the Vue videos and have coded along. So, explanations throughout the rest of this project are sparse. You should be able to do what is asked of you. If you don't understand how to do something, go back and check out the videos again. If I ask you to do something not covered in the videos, I'll give you more direction. But if it's in the videos, you should know how to do it!

v-text

1. Delete the content between the `<h1></h1>` tags so that the tag is empty.

2. Edit the opening H1 tag to use `v-text` to render the `listName`. Your page should still look like the previous screenshot.

3. Edit the returned data in the JS to access the albums array from the data returned by the server.

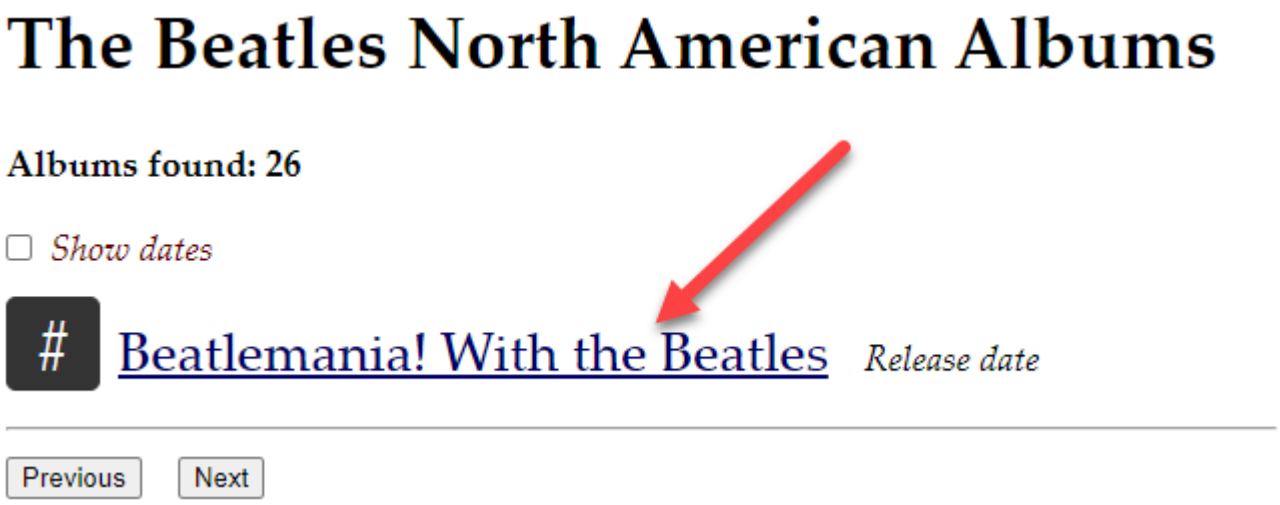
```
const ALBUMS_APP = {
  data() {
    return {
      listName: 'The Beatles North American Albums',
      albums: dataFromServer.albums
    }; // end return
  } // end data()
```

4. Delete *999* from the span tag so that the `<span></span>` tag is empty.

5. Edit the opening SPAN tag to use `v-text` to render the number of albums. Hint: this is simply the length of the albums array. And *albums* is now available in the Vue app: `albums.length`

6. Because `albums` is an array of objects, we can access each array member via its index. For example, `albums[0], albums[1], albums[2],` etc. We can then access the properties of each object like `albums[0].title, albums[0].released, albums[0].url, albums[1].title, albums[1].released`, etc. You'd need to look back at the JSON data file to learn the property names. Edit the opening <a> tag to include the following, which will display the title of the first album:

```
v-text="albums[0].title"
```



7. Change the `[0]` to other values (anything up to `[25]`) to see other album titles. If you try [26], you'll see the error message because there is no album at that index.

v-on and Array Iteration

Let's make the Previous | Next buttons work so we can scroll through the album titles. We'll need a counter to keep track of which array element we're on.

1. Initialize a counter as part of the returned data.

```
const ALBUMS_APP = {
  data() {
    return {
      listName: 'The Beatles North American Albums',
      albums: dataFromServer.albums,
      i: 0
    }; // end return
```

2. Editing the opening `<a>` tag to use the counter:

```
v-text="albums[i].title"
```

3. Edit the opening `button` tag for the Next button to call a function (method) when clicked.

```
<button v-on:click="moveForward">Next</button>
```

4. You know what – let's use shorthand instead. Change `v-on:click` to use shorthand. Remember how to do that? If not, check the videos.

5. Create the `moveForward` method. `data()` and `methods` should be siblings within `ALBUMS_APP`.
    a. Add a methods block after data()
    b. Create a method moveForward()
    c. Use a ternary operator to check if the counter `i` is less than the total albums, and if so, increment the counter. Otherwise, do nothing. This ensures the counter is not incremented beyond the last album.

```
        i,
      }; // end return
    }, // end data()
    methods: {
      moveForward() {
        this.i < dataFromServer.albums.length - 1 ? this.i++ : '';
      } // end moveForward
    } // end methods
  }; // end ALBUMS_APP
  Vue.createApp(ALBUMS_APP).mount('#app');
} // end fetchSuccess function
```

6. Test it. Every click of the Next button should display the next album title.

7. Add a second method called `moveBack`, making sure not to decrement it less than 0.

```
    }, // end a
    methods: {
      moveForward() {
        this.i < dataFromServer.albums.length - 1 ? this.i++ : '';
      }, // end moveForward
      moveBack() {
        this.i > 0 ? this.i-- : '';
      } // end moveForward
    } // end methods
  }; // end ALBUMS_APP
  Vue.createApp(ALBUMS_APP).mount('#app');
} // end fetchSuccess function
```

8. Edit the opening button tag for the Previous button to call `moveBack` when clicked. Test it. Both Previous and Next buttons should work to scroll you through the album titles. The Previous button should not work when the first title is display and the Next button shouldn't work when the final title is displayed.

v-bind

Currently, the album title is a link. But the `href` in the HTML points to #. We'll replace that with the url from the JSON data.

1. Edit the opening `<a>` tag to replace the current `href` with v-bind. While we're at it, delete the text inside the `<a></a>` tag because that being replaced by the existing `v-text`. Notice each attribute is on a new line for readability.

```
<a target="_blank"
   v-bind:href="albums[i].url"
   v-text="albums[i].title"></a>
```

2. Test it. Use the Previous | Next buttons to scroll through a bunch of titles and click the album title link. You should navigate to the Wikipedia page for that album.

3. You know what – let's use shorthand instead. Change `v-bind:href` to use shorthand. Remember how to do that? If not, check the videos.

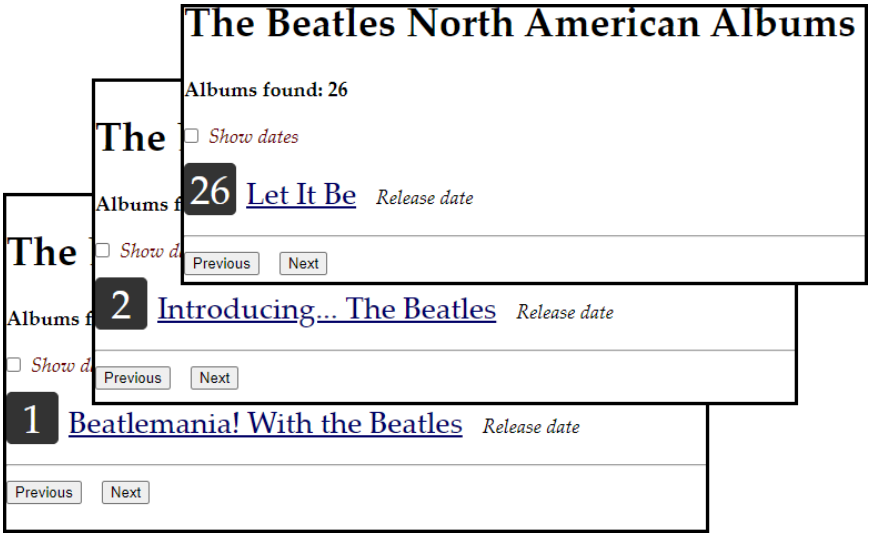
Album Counter

Let's add the current album number to the styled span to the left of the album title. Right now it just shows #. We already have a counter – `i` – that marks the current index. The first album is index 0, but we want it to show as #1.

1. Edit the `span id="number"` tag to render the 'current index plus one'. While we're at it, delete the placeholder text.

```
<span id="number" v-text="i+1"></span>
```

2. Test it. The number (1-26) should display next to the album title.



v-model

Lastly, let's hook up the checkbox to show the release date next to the album if the checkbox is checked.

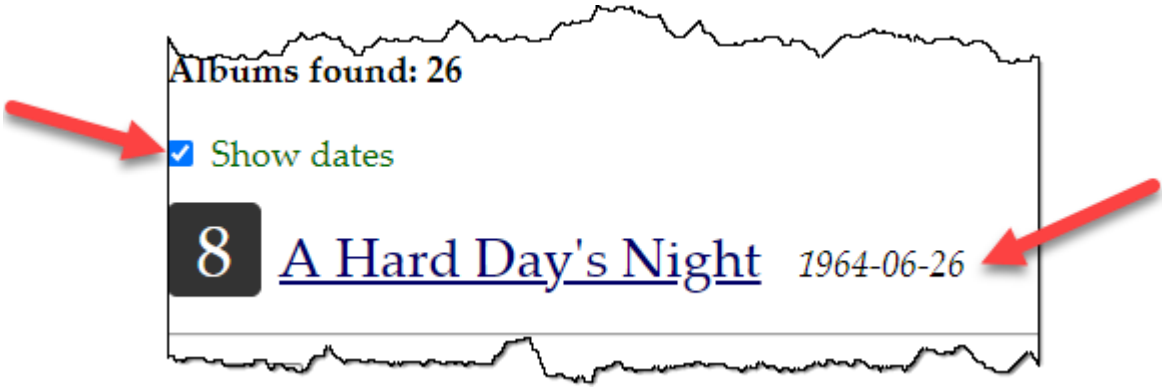
1. Set a flag in the return object to default a property – let's call it `showDates` – to `false`.

```
const ALBUMS_APP = {
  data() {
    return {
      listName: 'The Beatles North American Albums',
      albums: dataFromServer.albums,
      i: 0,
      showDates: false
    }; // end return
```

2. Edit the `input` tag to hook up the `showDates` property via `v-model`. Forgot how? Check the videos.

3. Edit the `span id="date"` tag. First, delete the placeholder content between the tags. Then, use a ternary operator within the opening tag to accomplish this logic:
   a. Use v-text to render a string
   b. If showDates is true, render the released property for the current album
   c. Otherwise, do nothing

4. Test it. If the box is checked, the date displays. If the box is cleared, the date disappears.



Summary

You should have a clear understanding of the objectives listed at the beginning of these instructions. The only thing not covered was `v-html` – which the same as `v-text` but allows html tags. Which one you use depends on the content you are rendering.

Submission

Name your project folder **vue-poc1-*flast***. Replace *flast* with your first initial and last name. For example, **vue-poc1-afehl**. ZIP it and submit via Moodle.