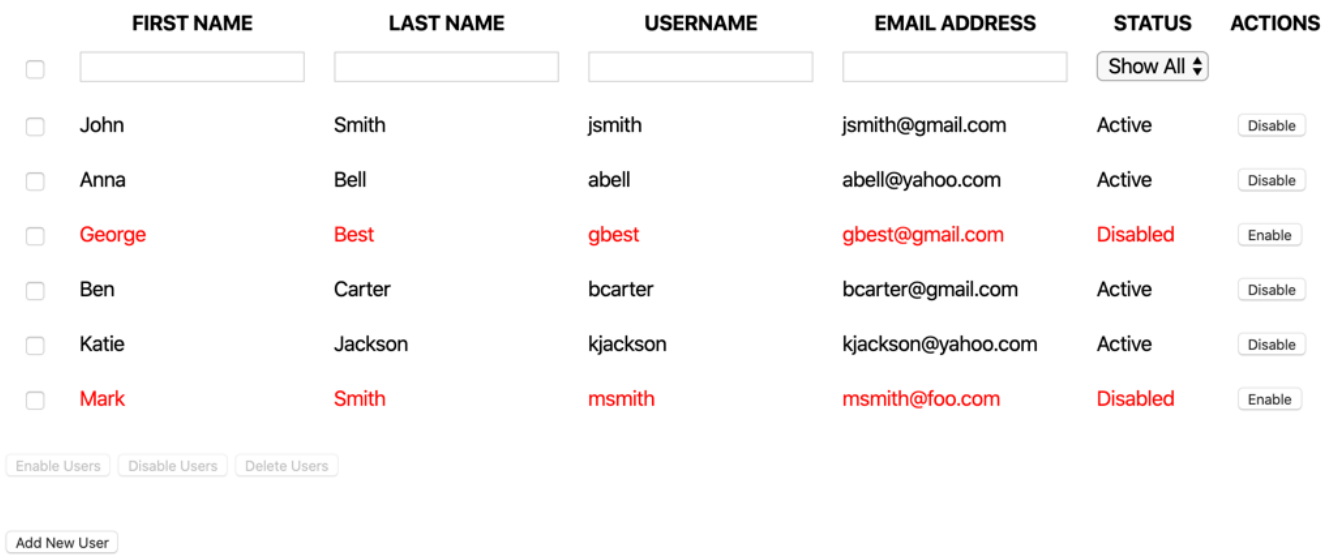
Vue.js Event Handling Exercise

In this exercise, you'll build on the previous exercise that asked you to display a list of users in a table. First, you'll start by creating a new user form to add users to the list. Second, you'll add the ability to select one or more users and then perform actions on them such as enable, disable, and delete. This is what the final application looks like:



> Before you begin the exercise, run `npm install` to install any dependencies.

Step One: Add a new user form

The markup for the new user form is already in the template. Complete the following tasks:

- Mark the form as hidden by default.
  - Use a property called `showForm`.
  - Toggle `showForm` with `v-on`.
  - Use `v-show` to show/hide form appropriately.
- Bind each form field to the correct property in the `newUser` object.
- Create a `saveUser` method that adds a new user to the `users` array on the form submission.
  - Get an id for the `newUser` object by calling the `getNextUserId()` method.
  - Clear the form after saving the user.

```
-------------------------------------
```

[Extra Hints:]

1. The v-show directive checks a boolean. Suppose that you had a boolean property called check. To use v-show, we would do:

```
<form id="frmAddNewUser" v-show="check==true">
```

OR

`<form id="frmAddNewUser" v-show="check">`

In this case, we need to make a new property in the data() section called showForm. If this property is in place, could we apply this to the v-show we just discussed?

2. We can associate a form element to a property in the data() section using v-model. This is what we mean by "bind each form field". In the data section you've been given the newUser object already.

In the form we have a field for first name. If we wanted to bind this field to the newUser object we would do the following:

`<input type="text" name="firstName" v-model="newUser.firstName" />`

You can now repeat this process for the remaining fields.

3. To complete the save functionality, since the form is bound to newUser, we can simply push the newUser object to the users array. Because of the binding, if we change the newUser object, we'll also change the form values.

To clear the form, we'll simply reset the object's properties to blank values:

```
this.newUser = {
  id: null,
  firstName: "",
  lastName: "",
  ........
  status: "Active"
}
```

Because of binding, the fields will clear as well.

> After you complete step one, run the end-to-end tests with the following command: `npm run test:e2e`, the tests under "Step One Tests, New User Form" pass.

Step Two: Add an enable/disable button in action column

In the "Action" column of the table, there's a button that enables or disables the user. Complete the following tasks:

- If the user status = 'Active', the button text displays 'Disable.'
- If the user status = 'Disabled', the button text displays 'Enable.'
- When you click the button, it calls a method `flipStatus()` and change the user's status from 'Active' to 'Disabled', or 'Disabled' to 'Active.'
    - The `flipStatus(id)` method takes the user ID as an argument.
    - You can use the user ID to find the user in the users array and change their status.

[Extra Hints:]

1. To set the text for the buttons dynamically we will have to write some code in the interpolation. A ternary statement would work just fine. In the users array, each object has a status property, which we can access with user.status.

So this is what the button code will look like:

```
<button class="btnEnableDisable">

{{ user.status === "Active" ? "something" : "something else"}}

</button>
```

You have to write in what "something" and "something else" is 😃.

2. We need to declare a method called flipStatus which takes in an id. After the method is declared we can can invoke it within the form when the button is clicked like so:

```
<button class="btnEnableDisable" v-on:click="flipStatus(user.id)">
```

The pseudo-code for this method goes something like this: Scan through the users array. If an object has an id that is the same as the id passed into the method, then for that object change the status to Disabled or Active.

```
-------------------------------------
```

> Once you complete this step, the tests under "Step Two Tests, Enable/Disable Action Button" pass.

Step Three: Disable, enable, and delete selected users

There are three buttons following the user listing table. Complete the following tasks associated with those buttons:

- Add a `selectedUserIDs` property that defaults to an empty array.
- Disable buttons when the `selectedUserIDs` array is empty.
  - Note: use a computed property named `actionButtonDisabled` for this.
- Add the user's ID to the `selectedUserIDs` array when user checks the checkbox.
  - Bind the checked value to if the user's ID is in the `selectedUserIDs` array.
- Enable Users
  - Sets the status of each selected user to `Active`.
  - Clears all checkboxes when action completes.
  - Method name: `enableSelectedUsers()`
- Disable Users
  - Sets the status of each selected user to `Disabled`.
  - Clears all checkboxes when action completes.
  - Method name: `disableSelectedUsers()`
- Delete Users
  - Deletes the user from the `users` array.
  - Clears all checkboxes when action completes.
  - Method name: `deleteSelectedUsers()`

> Tip: Depending how you retrieve the user ID, Vue may give it to you as a string even though it's a number. This may give you issues with comparison—try converting to a number before adding to `selectedUserIDs` if you have comparison issues.

[Extra Hints:]

1. selectedUserIDs can just be an empty array in the data section.
2. The first button should look like this:

```
<button v-bind:disabled="actionButtonDisabled">
Enable Users
</button>
```

The pseudo code for the computed property "actionButtonDisabled" goes something like this: if this.selectedUserIDs's length is 0 then return true, otherwise return false.

You can follow a similar process to get the remaining two buttons to enable or disable.

3. We have to add an on-click event to the add button, your first button will now look like this:

```
<button
v-bind:disabled="actionButtonDisabled">
v-action="enableSelectedUsers()"
Enable Users
</button>
```

The pseudo code for the method enableSelectedUsers goes something like this: loop through selectedUserIDs, for each id, find the matching id in the users array (it's fine to use two loops). When you find the matching object in the users array set its status to "Active".

You will write similar code for the Disable Users and Delete Users button.

> Once you complete this step, the tests under "Step Three Tests, Disable, Enable, and Delete Selected Users" pass.

Step Four: Select all checkbox

The "select all" checkbox is the checkbox on the first row with the filter inputs. Complete the following tasks for the "select all" checkbox:

- When clicked to "checked" state, set all boxes to checked and add all users to `selectedUserIDs`. When clicked again to "unchecked" state, set all boxes to unchecked and remove all users from `selectedUserIDs`.

- If some of the boxes are in a checked state, clicking "select all" selects all users. If clicked again, then it deselects all users.
- Check "Select all" when all individual checkboxes get checked.
- Add/remove the user IDs to/from the array depending upon the checked status of all checkboxes.

[Extra Hints:]

1. Identify the selectAll checkbox, it's in the tbody. We will add an event listener through a v-on:change directive (It's not a click!):

```
<input type="checkbox" id="selectAll" v-on:change="selectAll($event)"
```

Just like with vanilla JS, event.target can give us information on the element that initiated the event. Here is some pseudo-code for the selectAll method:

If event.target.checked is true, then clear the selectedUserIDs array, and add in every id from the users array. Otherwise, just clear the selectedUserIDs array.

You can always clear an array by just setting the array to an empty array.

At this point, all tests pass, as shown here: