**Form Validation**

[**Base Example**](https://vuejs.org/v2/cookbook/form-validation.html#Base-Example)

Form validation is natively supported by the browser, but sometimes different browsers will handle things in a manner which makes relying on it a bit tricky. Even when validation is supported perfectly, there may be times when custom validations are needed and a more manual, Vue-based solution may be more appropriate. Let’s begin with a simple example.

Given a form of three fields, make two required. Let’s look at the HTML first:

|  |
| --- |
| <form id="app" @submit="checkForm" action="https://vuejs.org/" method="post">    <p v-if="errors.length">  <b>Please correct the following error(s):</b>  <ul>  <li v-for="error in errors">{{ error }}</li>  </ul>  </p>    <p>  <label for="name">Name</label>  <input type="text" name="name" id="name" v-model="name">  </p>   <p>  <label for="age">Age</label>  <input type="number" name="age" id="age" v-model="age" min="0">  </p>   <p>  <label for="movie">Favorite Movie</label>  <select name="movie" id="movie" v-model="movie">  <option>Star Wars</option>  <option>Vanilla Sky</option>  <option>Atomic Blonde</option>  </select>  </p>   <p>  <input type="submit" value="Submit">   </p>  </form> |

Let’s cover it from the top. The <form> tag has an ID that we’ll be using for the Vue component. There’s a submit handler that you’ll see in a bit, and the action is a temporary URL that would point to something real on a server someplace (where you have backup server-side validation of course).

Beneath that there is a paragraph that shows or hides itself based on an error state. This will render a simple list of errors on top of the form. Also note we fire the validation on submit rather than as every field is modified.

The final thing to note is that each of the three fields has a corresponding v-model to connect them to values we will work with in the JavaScript. Now let’s look at that.

|  |
| --- |
| const app = new Vue({  el:'#app',  data:{  errors:[],  name:null,  age:null,  movie:null  },  methods:{  checkForm:function(e) {  if(this.name && this.age) return true;  this.errors = [];  if(!this.name) this.errors.push("Name required.");  if(!this.age) this.errors.push("Age required.");  e.preventDefault();  }  } }) |

Fairly short and simple. We define an array to hold errors and set null values for the three form fields. The checkForm logic (which is run on submit remember) checks for name and age only as movie is optional. If they are empty we check each and set a specific error for each. And that’s really it. You can run the demo below. Don’t forget that on a successful submission it’s going to POST to a temporary URL.

[**Using Custom Validation**](https://vuejs.org/v2/cookbook/form-validation.html#Using-Custom-Validation)

For the second example, the second text field (age) was switched to email which will be validated with a bit of custom logic. The code is taken from the StackOverflow question, [**How to validate email address in JavaScript?**](https://stackoverflow.com/questions/46155/how-to-validate-email-address-in-javascript). This is an awesome question because it makes your most intense Facebook political/religious argument look like a slight disagreement over who makes the best beer. Seriously - it’s insane. Here is the HTML, even though it’s really close to the first example.

|  |
| --- |
| <form id="app" @submit="checkForm" action="https://vuejs.org/" method="post" novalidate="true">    <p v-if="errors.length">  <b>Please correct the following error(s):</b>  <ul>  <li v-for="error in errors">{{ error }}</li>  </ul>  </p>    <p>  <label for="name">Name</label>  <input type="text" name="name" id="name" v-model="name">  </p>   <p>  <label for="email">Email</label>  <input type="email" name="email" id="email" v-model="email">  </p>   <p>  <label for="movie">Favorite Movie</label>  <select name="movie" id="movie" v-model="movie">  <option>Star Wars</option>  <option>Vanilla Sky</option>  <option>Atomic Blonde</option>  </select>  </p>   <p>  <input type="submit" value="Submit">   </p>  </form> |

While the change here is small, note the novalidate="true" on top. This is important because the browser will attempt to validate the email address in the field when type="email". Frankly it may make more sense to trust the browser in this case, but as we wanted an example with custom validation, we’re disabling it. Here’s the updated JavaScript.

|  |
| --- |
| const app = new Vue({  el:'#app',  data:{  errors:[],  name:null,  email:null,  movie:null  },  methods:{  checkForm:function(e) {  this.errors = [];  if(!this.name) this.errors.push("Name required.");  if(!this.email) {  this.errors.push("Email required.");  } else if(!this.validEmail(this.email)) {  this.errors.push("Valid email required.");   }  if(!this.errors.length) return true;  e.preventDefault();  },  validEmail:function(email) {  var re = /^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]+)\*)|(".+"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/;  return re.test(email);  }  } }) |

As you can see, we’ve added validEmail as a new method and it is simply called from checkForm. You can play with this example here:

[**Another Example of Custom Validation**](https://vuejs.org/v2/cookbook/form-validation.html#Another-Example-of-Custom-Validation)

For the third example, we’ve built something you’ve probably seen in survey apps. The user is asked to spend a “budget” for a set of features for a new Star Destroyer model. The total must equal 100. First, the HTML.

|  |
| --- |
| <form id="app" @submit="checkForm" action="https://vuejs.org/" method="post" novalidate="true">    <p v-if="errors.length">  <b>Please correct the following error(s):</b>  <ul>  <li v-for="error in errors">{{ error }}</li>  </ul>  </p>   <p>  Given a budget of 100 dollars, indicate how much  you would spend on the following features for the  next generation Star Destroyer. Your total must sum up to 100.  </p>   <p>  <input type="number" name="weapons" v-model.number="weapons"> Weapons <br/>  <input type="number" name="shields" v-model.number="shields"> Shields <br/>  <input type="number" name="coffee" v-model.number="coffee"> Coffee <br/>  <input type="number" name="ac" v-model.number="ac"> Air Conditioning <br/>  <input type="number" name="mousedroids" v-model.number="mousedroids"> Mouse Droids <br/>  </p>   <p>  Current Total: {{total}}  </p>   <p>  <input type="submit" value="Submit">   </p>  </form> |

Note the set of inputs covering the five different features. Note the addition of .number to the v-model attribute. This tells Vue to cast the value to a number when you use it. However, there is a bug with this feature such that when the value is blank, it turns back into a string. You’ll see the workaround below. To make it a bit easier for the user, we also added a current total right below so they can see, in real time, what their total is. Now let’s look at the JavaScript.

|  |
| --- |
| const app = new Vue({  el:'#app',  data:{  errors:[],  weapons:0,  shields:0,  coffee:0,  ac:0,  mousedroids:0  },  computed:{  total:function() {  // must parse cuz Vue turns empty value to string  return Number(this.weapons)+  Number(this.shields)+  Number(this.coffee)+  Number(this.ac+this.mousedroids);  }  },  methods:{  checkForm:function(e) {  this.errors = [];  if(this.total != 100) this.errors.push("Total must be 100!");  if(!this.errors.length) return true;  e.preventDefault();  }  } }) |

We set up the total value as a computed value, and outside of that bug I ran into, it was simple enough to setup. My checkForm method now just needs to see if the total is 100 and that’s it. You can play with this here:

[**Server-side Validation**](https://vuejs.org/v2/cookbook/form-validation.html#Server-side-Validation)

In my final example, we built something that makes use of Ajax to validate at the server. The form will ask you to name a new product and will then check to ensure that the name is unique. We wrote a quick [**OpenWhisk**](http://openwhisk.apache.org/) serverless action to do the validation. While it isn’t terribly important, here is the logic:

|  |
| --- |
| function main(args) {   return new Promise((resolve, reject) => {   // bad product names: vista, empire, mbp  let badNames = ['vista','empire','mbp'];  if(badNames.includes(args.name)) reject({error:'Existing product'});  resolve({status:'ok'});   });  } |

Basically any name but “vista”, “empire”, and “mbp” are acceptable. Ok, so let’s look at the form.

|  |
| --- |
| <form id="app" @submit="checkForm" method="post">    <p v-if="errors.length">  <b>Please correct the following error(s):</b>  <ul>  <li v-for="error in errors">{{ error }}</li>  </ul>  </p>   <p>  <label for="name">New Product Name: </label>  <input type="text" name="name" id="name" v-model="name">  </p>   <p>  <input type="submit" value="Submit">   </p>  </form> |

There isn’t anything special here. So let’s go on to the JavaScript.

|  |
| --- |
| const apiUrl = 'https://openwhisk.ng.bluemix.net/api/v1/web/rcamden%40us.ibm.com\_My%20Space/safeToDelete/productName.json?name=';  const app = new Vue({  el:'#app',  data:{  errors:[],  name:''  },  methods:{  checkForm:function(e) {  e.preventDefault();  this.errors = [];  if(this.name === '') {  this.errors.push("Product name is required.");  } else {  fetch(apiUrl+encodeURIComponent(this.name))  .then(res => res.json())  .then(res => {  if(res.error) {  this.errors.push(res.error);  } else {             // redirect to a new URL, or do something on success  alert('ok!');  }  });  }  }  } }) |

We start off with a variable representing the URL of the API that is running on OpenWhisk. Now look at checkForm. In this version, we always prevent the form from submitting (which, by the way, could be done in the HTML with Vue as well). You can see a basic check on this.name being empty, and then we hit the API. If it’s bad, we add an error as before. If it’s good, right now we do nothing (just an alert), but you could navigate the user to a new page with the product name in the URL, or do other actions as well. You can run this demo below: