**Consume Remote API Data Via HTTP In A Vue.js Web Application**

When building a modern web application, chances are that you’ll need to consume data from some remote resource, whether it be one that you’ve built or something someone else built. Sending HTTP requests is one of the more popular ways to send data from client facing applications to RESTful API backends.

We’re going to see how to send HTTP requests within a Vue.js web application, using a variety of techniques such as those found in the popular axios and vue-resource libraries.

## Getting Started with a New Vue.js Project

For simplicity, we’re going to create a fresh Vue.js project and integrate each of the axios and vue-resource libraries into it. Assuming you’ve got the Vue CLI installed, execute the following:

vue init webpack http-project

When prompted, fill in the project information. It does not matter if you choose standalone (runtime and compiler) project vs the runtime-only project. Just say no to all the testing features and the routing. This is a very bare bones project that we’re going to work with.

After the project has been created, execute the following:

cd http-project

npm install

At this point we should be ready to start development.

## Make HTTP Requests to Remote Web Services with axios

One of the more popular HTTP libraries out there is called axios. It is dead simple and very lightweight, which makes it a great solution for any Vue.js project.

To include axios in your project, execute the following:

npm install axios --save

Going forward, all our development will happen in the **src/components/HelloWorld.vue** file that was included when we scaffolded the project with the Vue CLI.

Without getting too far ahead of ourselves, open the project’s **src/components/HelloWorld.vue** file and include the following:

<template>

<div class="hello">

<h1>Your IP is {{ ip }}</h1>

<input type="text" v-model="input.firstname" placeholder="First Name" />

<input type="text" v-model="input.lastname" placeholder="Last Name" />

<button>Send</button>

<br />

<br />

<textarea>{{ response }}</textarea>

</div>

</template>

<script>

import axios from "axios";

export default {

name: 'HelloWorld',

data () {

return {

ip: "",

input: {

firstname: "",

lastname: ""

},

response: ""

}

},

mounted() { },

methods: { }

}

</script>

<style scoped>

h1, h2 {

font-weight: normal;

}

ul {

list-style-type: none;

padding: 0;

}

li {

display: inline-block;

margin: 0 10px;

}

a {

color: #42b983;

}

textarea {

width: 600px;

height: 200px;

}

</style>

In the above code, notice that we’ve initialized a few variables in the data method of the <script> tags. These variables are bound to components found in the <template>. For example, the ip variable will be printed, the input object will have a two-way data binding with our form, and the response variable will be printed.

Within the <script> tag you’ll also notice the following:

import axios from "axios";

This is all it takes to start using the axios library in our code.

Rather than creating our own set of RESTful APIs, let’s go ahead and leverage one that is already available and designed for testing. We’re going to make use of httbin.

The first thing we do when it comes to HTTP will be obtaining our IP address when the application loads. We can send a GET request over HTTP to httpbin to find this information.

Take a look at the following JavaScript code:

<script>

import axios from "axios";

export default {

name: 'HelloWorld',

data () {

return {

ip: "",

input: {

firstname: "",

lastname: ""

},

response: ""

}

},

mounted() {

axios({ method: "GET", "url": "https://httpbin.org/ip" }).then(result => {

this.ip = result.data.origin;

}, error => {

console.error(error);

});

},

methods: { }

}

</script>

Using axios we can send a request and work with a promise. If the request is successful, we will assign our ipvariable the value of our origin IP found in the response.

Not so bad right?

Now let’s create a method for sending a POST request over HTTP based on our form information. Take a look at the following JavaScript code:

<script>

import axios from "axios";

export default {

name: 'HelloWorld',

data () {

return {

ip: "",

input: {

firstname: "",

lastname: ""

},

response: ""

}

},

mounted() {

axios({ method: "GET", "url": "https://httpbin.org/ip" }).then(result => {

this.ip = result.data.origin;

}, error => {

console.error(error);

});

},

methods: {

sendData() {

axios({ method: "POST", "url": "https://httpbin.org/post", "data": this.input, "headers": { "content-type": "application/json" } }).then(result => {

this.response = result.data;

}, error => {

console.error(error);

});

}

}

}

</script>

We’ve created a sendData method that sends the input object as payload data. Because we’re sending an object, we need to specify via HTTP headers that the content is of JSON format.

Upon success, we set the response variable to whatever the result is. However, we need a way to call the sendData method. Head back into the <template> block and change the markup to look like the following:

<template>

<div class="hello">

<h1>Your IP is {{ ip }}</h1>

<input type="text" v-model="input.firstname" placeholder="First Name" />

<input type="text" v-model="input.lastname" placeholder="Last Name" />

<button v-on:click="sendData()">Send</button>

<br />

<br />

<textarea>{{ response }}</textarea>

</div>

</template>

We’ve added an v-on:click event to the button. This will call the sendData method and the bound form data will be used.

There are many more things that can be done with axios. To see the true power that axios offers, check out the official documentation.

## Make HTTP Requests to Remote Web Services with vue-resource

The axios library isn’t the only way to make HTTP requests. The library vue-resource was once packaged into Vue.js, but was since separated into another project. This doesn’t make the project deprecated, just externally maintained.

Before we can use the vue-resource library, it must be downloaded. From the command line, execute the following:

npm install vue-resource --save

The setup to use the vue-resource library is slightly different than what we had to do for the axios library. Open the project’s **src/main.js** file and include the following:

import Vue from 'vue'

import App from './App'

import VueResource from 'vue-resource';

Vue.use(VueResource);

Vue.config.productionTip = false

new Vue({

el: '#app',

render: h => h(App)

})

Notice that we’ve imported the library and told Vue to use it. By doing this, we’ll be able to use the $http service in Vue.js.

Head back into the project’s **src/components/HelloWorld.vue** file. We’re going to only alter the <script> block. Everything that we had previously created for axios can remain the same.

<script>

export default {

name: 'HelloWorld',

data () {

return {

ip: "",

input: {

firstname: "",

lastname: ""

},

response: ""

}

},

mounted() {

this.$http.get("https://httpbin.org/ip").then(result => {

this.ip = result.body.origin;

}, error => {

console.error(error);

});

},

methods: {

sendData() {

this.$http.post("https://httpbin.org/post", this.input, { headers: { "content-type": "application/json" } }).then(result => {

this.response = result.data;

}, error => {

console.error(error);

});

}

}

}

</script>

Things have only changed slightly between our vue-resource version and our axios version. Now we can call the $http.get and $http.post methods to do business. How we structure them is similar to the previous and likewise with the response.