Connecting to Servers via HTTP using vue-resource

A big feature in Vue.js. You need to fetch and store data on external servers. SPA usually use AJAX to get resource behind the scene. There are many external AJAX libraries that we can use within Vue.js application.

Setup vue-resource

To install **vue-resource** into current project. Issue the following command. Find more detail about this at *https://github.com/pagekit/vue-resource*.

```
# npm install --save vue-resource
```

After that, in **main.js**, we just need to import and register the plugin with **Vue.use()** method.

```
// main.js
...
import VueResource from 'vue-resource';

Vue.use(VueResource);
...
```

Creating an Application and Setting up a Server

Now we need a server to store and fetch data. We can use **Google Firebase** service (*https://firebase.google.com*). After creating a new

project, go to the **Database->Rules** and change the rule to allow read and write permission and publish. Note that the following setting is only for **testing** not for production.

```
// Firebase's Database Rules:
{
  "rules": {
    ".read": "true",
    ".write": "true"
  }
}
```

Go back to the **DATA** tab and we will see the **URL** for us to access. For example:

```
https://vuejs-http-xxxxx.firebaseio.com/
```

Now we can create an input form in the **App.vue** as following:

```
// App.vue
<template>
...
  <h1>Http</h1>
  <div class="form-group">
      <label>Username</label>
      <input type="text" class="form-control" v-
model="user.username">
  </div>
  <div class="form-group">
      <label>Mail</label>
      <input type="text" class="form-control" v-
model="user.email">
  </div>
```

```
  <button class="bth bth-primary"
@click="submit">Submit</button>
...
</template>

<script>
...
...
</script>
```

Sending a POST request

To send a request to **firebase**, we can use **$http** method (due to vue-resource) to do so. First we try using **post** method to send **user** object. Note that the database name should be in the pattern of **[name].json**.

```
// App.vue

<script>
    export default {
        data() {
            return {
                user: {
                    username: '',
                    email: ''
                }
            }
        },
        methods: {
            submit() {
                this.$http.post('https://vuejs-http-
6b751.firebaseio.com/users.json', this.user)
                .then(response => {         // define
how to deal with the response
                    console.log(response);
                }, error => {               // define
```

```
  how to deal with error
                      console.log(error);
                  });
              }
          }
      }
  </script>
```

Sending a GET request

Now let us try fetching data from firebase. First create a template to show the fetched data.

First, we create a template to display the data as following:

```
// App.vue
<template>
...
  <!-- show fetching data in a list -->
  <hr>
  <button class="btn bth-primary"
@click="fetchData">Get Data</button>
  <br><br>
  <ul class="list-group">
      <li class="list-group-item" v-for="u in users"
:key="u.email">{{ u.username }} - {{ u.email }}</li>
  </ul>
...
</template>
```

Next, we create a **fetchData()** method and use the **$http.get()** method to make a request. The Vue component code is modified as following:

```
<script>
```

```
    export default {
        data() {
            return {
                user: {
                    username: '',
                    email: ''
                },
                users: []
            }
        },
        methods: {
            ...
            fetchData() {
              this.$http.get('https://vuejs-http-
6b751.firebaseio.com/users.json')
                .then(response => {
                    const data = response.json();
                    console.log(data);      // this
prints the promise object NOT the data object
                }, error => {
                    console.log(error);
                });)
            }
        }
    }
</script>
```

The **response** from the **$http.get()** method is a **promise** NOT the data because this is a **asynchronous process** that does not return the data right away. To get the real data, an array of users, we have to return the **response.json()** and define what to do with the data using **.then()** method.

```
<script>
...
  fetchData() {
```

```
    this.$http.get('https://vuejs-http-
6b751.firebaseio.com/users.json')
      .then(response => {
        return response.json();           // return
an a javascript object
      })
      .then(data => {                     // define
what to do with the returned javascript object
        const resultArray = [];
        for (let key in data) {
            resultArray.push(data[key]);
        }
        this.users = resultArray;         // trigger
Vue.js to update the DOM
      });
  }
...
</script>
```

Configuring vue-resource globally

If we do not want to repeat the URL evertime, we can use **Vue.http.options** in the **main.js** to specify **based URL** for our requests as following:

```
// main.js
...
Vue.http.options.root = 'https://vuejs-http-
6b751.firebaseio.com/users.json';
...
```

Now all requests will be sent to the specified URL and , in the methods, we can simply **append** the based URL.

```
// App.vue
<script>
...
  submit() {
      this.$http.post('', this.user)      // or
('/[service_name]', this.user)
      ...;
  },
  fetchData() {
      this.$http.get('')
      ...;
  }
...
</script>
```

We can setup many things using **Vue.http.options** object such as header(). We can learn more about Vue.js **api** from *https://github.com/pagekit/vue-resource/tree/develop/docs*.

Intercepting Requests

Interceptors can be defined globally and are used for **pre-** and **postprocessing** or a request **Vue.http.interceptions** an array of function we can execute on each request. In the **main.js** add the following code:

```
// main.js
...
// intercepting POST requests and change them to PUT
Vue.http.interceptors.push((request, next) => {
  console.log(request);
  if(request.method == 'POST') {
    request.method = 'PUT';
  }
  next();
});
```

```
...
```

Find more detail at *https://github.com/pagekit/vue-resource/blob/develop/docs/http.md*

Intercepting Responses

To intercept the response, we can define what to do in the **next()** method. For example, we can override the **response** in the **fetchData** method:

```
// main.js
...
// intercepting POST requests and change them to PUT
Vue.http.interceptors.push((request, next) => {
  console.log(request);
  if(request.method == 'POST') {
    request.method = 'PUT';
  }
  next(response => {      // this will intercept all
reponses
    console.log(response.body);
    response.json = () => { return {messages:
response.body } }   // replace with an object with a
key
  });
});
...
```

Now the **response.json** is a javascript object with 'message' as a key. We need a key in order to display it in the template. Normally you would want to use intercepter to change something like **header**.

Where the RESOURCE in vue-resource comes from

There is a feature to setup resource (not only HTTP request). In **main.js**, let's change the **based URL** by removing the resource (*/users.json*).

```
// main.js

// Vue.http.options.root = 'https://vuejs-http-
6b751.firebaseio.com/users.json';
Vue.http.options.root = 'https://vuejs-http-
6b751.firebaseio.com';
```

We can create a **resouce** by creating new **resource** data property and using the **created()** lifecycle hook to initialized the property with **this.$resource** object.

```
// App.vue
  data() {
    return {
      ...
      resource: {}     // an empty object for storing
resources
    }
  },
  methods: {
      submit() {
        this.resource.save({}, this.user);  // inside
{} are the parameters we can pass
      },
      fetchData() {
        this.$http.get('user.json')         // make a
GET request using $http.get() with new based URL
      ...;
  }
      ...
  },
  created() {
```

```
      this.resource = this.$resource('users.json');
// setup local resource to 'users.json'
  }
```

Find more detail at *https://github.com/pagekit/vue-resource/blob/develop/docs/resource.md*

Creating Custom Resources

Sometimes we might want to use custom resource rather than the normal resource. This done by creating a **custom resource object** and assigned it to the **this.$resource**.

```
// App.vue
  data() {
    return {
      ...
      resource: {}     // an empty object for storing
resources
    }
  },
  methods: {
      submit() {
        this.resource.saveAlt(this.user);
      },
      ...
  },
  created() {
      const customAction = {
        saveAlt: { method: 'POST', url:
'alternative.json'}   // make POST request to
different resource
      }
      this.resource = this.$resource('users.json',
{}, customAction);
  }
```

This will create (or update) a user in the **alternative.json** database in firebase.

Resources vs. normal HTTP requests

The resource is just an alternative to HTTP request.

Template URLs

Basically, it is a *variable* in URL. You can define a resource template by using sort of 'interpolation' in URL. The following example allows user to specify the **database** to fetch data. First, we need to add text input in our template and bind its value with the **node** data property.

```
// App.vue
<template>
...
  <!-- a new text input -->
  <input type="text" class="form-control" v-
model="node">
  <br><br>
  <button class="btn bth-primary"
@click="fetchData">Get Data</button>
  <br><br>
  <ul class="list-group">
      <li class="list-group-item" v-for="u in users"
:key="u.email">{{ u.username }} - {{ u.email }}</li>
  </ul>
...
</template>

<script>
...
  data() {
    return {
      ...
```

```
      resource: {},     // an empty object for
storing resources
      node: 'users'     // default database:
'users.json'
    }
  },
  methods: {
    ...
    fetchData() {
      // replace '{node}.json' with this.node
      this.resource.getData({node: this.node})    //
using resource template
      .then(response => {                         //
the same as before
          return response.json();
      })
      .then(data => {
          const resultArray = [];
          for (let key in data) {
              resultArray.push(data[key]);
          }
          this.users = resultArray;
      });
    }
  }
  created() {

      const customAction = {
          saveAlt: { method: 'POST', url:
'alternative.json'},
          getData: { method: 'GET'}
// create getData object
      }
      this.resource = this.$resource('{node}.json',
{}, customAction);   // passing data fields
  }
...
</script>
```