Lab 5: Interacting with APIs using VueJS

Now that we understand the basics of Vue, we can start using it to interact with the APIs that we have created.

1 Exercise 1: Initial setup

We are going to build a front end application for the chat app API that we wrote in lab 3. Start by running the API that you created for lab 3.

Note: If your API is hosted at a different location to your client application (i.e. a resource has to make a cross-origin HTTP request to a different domain, protocol, or port) then you will need to add CORS support to the API code. This allows your API to accept requests from different domains. One method to add CORS support is to add the below code into your express config file. More information about CORS can be found at: https://enable-cors.org/index.html.

```
app.use(function(req, res, next) {
        res.header("Access-Control-Allow-Origin", "*");
        res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With,
Content-Type, Accept");
       res.header("Access-Control-Allow-Methods", "GET, POST, PUT, DELETE");
        next();
});
```

2 Exercise 2: Interacting with an API: User functionality

In this exercise, we will create a Vue application that calls our API from lab 3, implementing all of the user features. Let's first remind ourselves of the API calls that we can make regarding the users.

URI	Method	Action
/api/user	GET	List all users
/api/user/:id	GET	List a single user
/api/user	POST	Add a new user
/api/user/:id	PUT	Edit an existing user
/api/user/:id	DELETE	Delete a user

2.1 Exercise 2.1: List all users

We will start by simply getting a list of all users and displaying them on the screen.

1. Create a new directory called ‘lab 5’
2. Use npm to install vue and vue-resource
3. Create a index.html file and a app.js file
4. Add the boilerplate code to the HTML file like in the previous lab. Although instead of using a CDN, use your installed copy of Vue. Also import Vue Resource into your HTML file.

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Application</title>
  </head>
  <body>
    <div id="app">

    </div>


    <script src="./node_modules/vue/dist/vue.js"></script>
    <script src="./node_modules/vue-resource/dist/vue-resource.js"></script>
    <script src="./app.js"></script>
  </body>
</html>
```

5. In your app.js file, create a Vue instance and add the following boilerplate code. **Note**: we have added a ‘mounted’ function. This function will run before the page is loaded.

```
Vue.http.options.emulateJSON = true;
new Vue({
    el: '#app',

    data: {

    },

    mounted: function() {

    },

    methods: {

    }
});
```

Note: setting Vue resource to emulate JSON ensures that JSON objects are sent out and read into our client applications. This is required for interacting with the assignment one API’s.

6. Add a variable inside the data object called ‘users.’ This should be an empty list.

7. Add the following method to your method object. This method uses Vue Resource to query the API. If the API call is successful then the users variable is populated with a list of users.

```
getUsers: function() {
  this.$http.get('http://url-to-your-api.com/api/users')
    .then(function(response){
      this.users = response.data;
    }, function(error){
      console.log(error);
    });
}
```

8. Call getUsers inside the mounted function

```
mounted: function() {
    this.getUsers();
},
```

9. In your HTML, use the v-for directive to list the users

```
<ol>
  <li v-for="user in users">
    {{ user.username }}
  </li>
</ol>
```

10. Run in your browser and test.

2.2 Exercise 2.2: Adding a new user

Next we want to write the functionality that allows us to add a new user.

1. In app.js, add a variable to the data object called ‘username.’ Initially set this to an empty string.
2. Add a method to your app.js file called ‘addUser,’ copy into it the code below.

```
addUser: function() {
    if (this.username === "") {
        alert("Please enter an username !");
    } else {
        this.$http.post('http://localhost/api/users', {
            "username": this.username
        });
    }
},
```

3. In your HTML page, create a form. Use the ‘v-on:submit’ directive to set the action to be the new method. User the ‘v-model’ directive to bind the input to the username variable created earlier.

```
<h2>Add a new user</h2>
<form v-on:submit="addUser()">
  <input v-model="username" placeholder="Username" />
  <input type="submit" value="Add" />
</form>
```

4. Open in your browser and test

2.3 Exercise 2.3: Deleting a user

1. Add the following method to your app.js. **Note**: On the successful deletion of the user from the database using the API, we here loop through our list of users to remove the user which we have just deleted. This allows us to remove the user from the DOM without having to refresh the page.

```
deleteUser: function(user) {
    this.$http.delete('http://localhost/api/users/' + user.user_id)
    .then(function(response){
        var tempid = user.user_id;

        for(var i = 0; i < this.users.length; i++){
            if(tempid == this.users[i].user_id){
               this.users.splice(i, 1);
            }
        }

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

2. In the HTML page, edit the list of users so that they each also list a ‘delete’ button.

```
<ul>
  <li v-for="user in users">
    <p>
      {{ user.username }}
      <button v-on:click="deleteUser(user)">Delete</button>
    </p>
  </li>
</ul>
```

3. Open in your browser and test

2.4 Exercise 2.4: Editing a user

Using the code examples from the previous parts of exercise two and the ‘$http.put’ method, implement code to edit a particular user. Don’t worry about imperfections, as next week's lab will look at how to structure your Vue applications. A simple implementation example is provided in the solution code for the lab.

3 Exercise 3: Implement the rest of the application - optional

This last exercise is optional. Carry on working until you are comfortable with the concepts covered.

Using the details and API specification given in Lab 2, create the rest of the chat application. You may implement the different aspects of the application in different pages, or contain it all to just the one page (index.html). Add styling to your application to make it responsive (using the HTML/CSS pre-lab for help).