CWEB280 -wk9 – LO6 – Web Page UI

# Vue JS - <https://vuejs.org/v2/guide/#What-is-Vue-js>

Vue is a progressive framework for building user interfaces.

* Vue.js is focused on the view layer only.
* Vue.js can also build Single-Page Applications when combined tooling and supporting libraries.
* A Vue.js is make up of 2 parts
  1. **The Vue instance**: a Javascript Object with separate sections(objects) for data (state), methods, computed properties and lifecycle hooks that are used to add functionality to the content within the HTLM element.
  2. **Declarative HTML Rendering**: a single element (HTML tag) becomes the container where all the Vue.JS directives, bindings, event handlers and models are used to produce JavaScript that renders elements directly to the DOM

## The Vue Instance - <https://vuejs.org/v2/guide/instance.html>

Every Vue application starts by creating a new Vue instance with the Vue function:

var vm = new Vue({

// options

})

When you create a Vue instance, you pass in an options object. You can use these options to create your desired behavior.

## Vue Instance Data Options - <https://vuejs.org/v2/api/#Options-Data>

Some of the options include:

[data](https://vuejs.org/v2/api/#data) - The data object for the Vue instance. Vue will recursively convert its properties into getter/setters to make it “reactive”. The object must be plain/native objects. Any prototype properties are ignored. The data properties are most often muted by the methods (see section below) or by the v-model directive

var vm = new Vue({

data: {

a: 'some string', // string literals

b: 2, // numbers

c: [1,2,3,4], // array

d: {a:1,b:true},// JS object

e: false, // boolean

},

})

[methods](https://vuejs.org/v2/api/#methods) - Methods to be mixed into the Vue instance. You can access these methods directly on the VM instance, or use them in directive expressions. Methods are mostly used to mutate the data properties  
DO NOT use double arrow notation to define a method (plus: () => this.a++) otherwise the ‘this’ keyword will not refer to the Vue instance

var vm = new Vue({

data: { a: 1 },

methods: {

incrementA() {

this.a++;

}

}

})

[computed](https://vuejs.org/v2/api/#computed) - Computed properties to be mixed into the Vue instance. Computed properties are cached, and only re-computed on reactive dependency changes. They act like self-mutating data properties (kind of like the data and methods sections combined)

DO NOT use double arrow notation to define a method (plus: () => this.a++) otherwise the ‘this’ keyword will not refer to the Vue instance

var vm = new Vue({

data: { a: 1 },

computed: {

// get only

aDouble() {

return this.a \* 2

},

// both get and set

aPlus: {

get() {

return this.a + 1

},

set(v) {

this.a = v - 1

}

}

}

})

## Vue Instance DOM Options - <https://vuejs.org/v2/api/#Options-DOM>

el - Provide the Vue instance an existing DOM element to mount on. It can be a CSS selector string or an actual HTMLElement.

var vm = new Vue({

el: '#app', // css selector to the element with id="app"

data: { a: 1 },

})

## The Vue Instance LifecycleVue Instance Lifecycle Hooks - <https://vuejs.org/v2/api/#Options-Lifecycle-Hooks>

Some of the life cycle hooks include:

<mounted> - Called after the instance has been mounted, where el is replaced by the newly created vm.$el. This hook is not called during server-side rendering.

var vm = new Vue({

data: { a: 1 },

methods: {

incrementA() {

this.a++;

}

}

mounted() {

this.incrementA();

}

})

# Use Vue.js in a simple HTML page

Before we create a complete Vue.js project we aew going to dip out toes into a simple html page that uses Vue.js libraries.

In the lo4serveapi project root create a new html file

\temporary-student-ui.html

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">  
 <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootswatch/4.3.1/solar/bootstrap.min.css">  
 <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.11.2/css/all.css">  
 <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-vue/dist/bootstrap-vue.min.css">  
 <title>Temporary Student UI</title>  
</head>  
<body>  
  
  
<!-- This div is rendered by Vuejs - Allows for Vuejs directives and componets -->  
<div class="container-fluid" id="app">  
 <b-jumbotron header="Temporary Student UI" lead="Quick and Dirty">  
 </b-jumbotron>

<!— OPTIONAL: used to DEBUG the data properties/variables by rendering them on the page-->  
 <b-alert variant="secondary" dismissible show>  
 <pre>  
DATA:  
{{$data}}  
 </pre>  
 </b-alert>

</div>

<!— IMPORT: JS Libraries from the internet-->  
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>  
<script src="https://cdn.jsdelivr.net/npm/portal-vue/dist/portal-vue.umd.js"></script>  
<script src="https://cdn.jsdelivr.net/npm/bootstrap-vue/dist/bootstrap-vue.js"></script>  
<script>  
var vm = new Vue({  
 el: '#app', // css selector to find the div with id="app"  
 data:{ // essentially the app state - all the data currently used by the app for rendering  
 students:[], // contains all the student objects used to display in a table  
 },  
 methods:{

},  
 computed:{ // this section acts like dynamic data properties/variables  
  
 },  
 mounted(){ // the hook for the part of the lifecycle when the app is started and mounts the app element

}  
 });  
</script>  
</body>  
</html>

NOTICE: The b-jumbotron and the b-alert tags are “components” created by a 3rd party library called “boostrap-vue”. We will learn more about components and boostrap-vue later in the course

## Graphical user interface, application Description automatically generatedUsing Webstorm’s built in webserver

1. In webstorm, right click temporary-student-ui.html and select Run
2. This will open the web browser to   
   http://localhost:63342/lo4serverapi/temporary-student-ui.html?\_ijt=7d8le6rbokk1ek0borkb2hkrsc
3. There is also a new configuration

A screenshot of a computer

Description automatically generated

1. DO NOT mix up the **start** configuration (launches typeorm’s webserver) and the temporary-student-ui.html configuration (launches webstorm’s built in webserver)   
   The lo4serverapi project will make use of 2 different webservers

# Declarative Rendering

Vue.js makes use an html element as a container for its rendering code.

Example: using 3rd party components in a div with id="app"

<!-- This div is rendered by Vuejs - Allows for Vuejs directives and componets -->  
<div class="container-fluid" id="app">  
 <b-jumbotron header="Temporary Student UI" lead="Quick and Dirty">  
 </b-jumbotron>

<!— OPTIONAL: used to DEBUG the data properties/variables by rendering them on the page-->  
 <b-alert variant="secondary" dismissible show>  
 <pre>  
DATA:  
{{$data}}  
 </pre>  
 </b-alert>

</div>

## Vue.js Directives – <https://vuejs.org/v2/api/#Directives>

A directive is some special token in the markup that tells the library to do something to a DOM element.

* directives can bind themselves to a data property on the Vue instance
* directive can also bind to an expression which is evaluated in the context of the instance
* When underlying data property or expression changes the directives automatically update the rendered content

Some directives include:

[v-if](https://vuejs.org/v2/api/#v-if) - Conditionally render the element depending if expression is true or false. The element and its children are destroyed and re-constructed during toggles.

[v-else](https://vuejs.org/v2/api/#v-else) - Does not expect an expression BUT the previous sibling element must have v-if or v-else-if.

<div v-if="Math.random() > 0.5">

Now you see me

</div>

<div v-else>

Now you don't

</div>

[v-else-if](https://vuejs.org/v2/api/#v-else-if)  - The previous sibling element must have v-if or v-else-if. Denotes the “else if block” for v-if and can be chained.

<div v-if="type === 'A'"> A </div>

<div v-else-if="type === 'B'"> B </div>

<div v-else-if="type === 'C'"> C </div>

<div v-else> Not A/B/C </div>

[v-for](https://vuejs.org/v2/api/#v-for) – Loops through and array, object, number, string or Iterable. Renders the element multiple times based on the source data. Best to provide an ordering hint with the key special attribute

<div v-for="item in items" :key="item.id">

{{ item.text }}

</div>

Alternatively, you can also specify an alias for the index (or the key if used on an Object):

<div v-for="(item, index) in items"></div>

<div v-for="(val, key) in object"></div>

<div v-for="(val, name, index) in object"></div>

[v-on (shorthand @)](https://vuejs.org/v2/api/#v-on) – Connects an event handler to the event. When listening to native DOM events, the method receives the native event as the only argument. If using inline statement, the statement has access to the special $event property: v-on:click="handle('ok', $event)".

<button v-on:click="doThis"></button>

<button v-on:click="doThat('hello', $event)"></button>

<!—short hand using @ symbol instead of v-on: -->

<button @click="doThis"></button>

<!-- stop propagation -->

<button @click.stop="doThis"></button>

<!-- prevent default -->

<button @click.prevent="doThis"></button>

<!-- prevent default without expression -->

<form @submit.prevent></form>

<!-- key modifier using keyAlias -->

<input @keyup.enter="onEnter">

[v-bind (shorthand :)](https://vuejs.org/v2/api/#v-bind) - Dynamically bind one or more attributes, or a component prop to an expression. When used without an argument, can be used to bind an object containing attribute name-value pairs.

<!-- bind an attribute -->

<img v-bind:src="imageSrc">

<!-- shorthand -->

<img :src="imageSrc">

<!-- with inline string concatenation -->

<img :src="'/path/to/images/' + fileName">

[v-model](https://vuejs.org/v2/api/#v-model) – Only works with: <input>, <select> and <textarea> tags. Creates a two-way binding on a form input to a data property or other form inputs

<input type="text" v-model="student.firstName" />

<input type="radio" v-model="pick" :value="a">

<input type="checkbox" v-model="agree" true-value="yes" false-value="no" />

<select v-model="province">

<option value="AL">Alberta</option>

</select>

<textarea v-model="message" placeholder="add multiple lines"></textarea>

# Display Student properties in an HTML Table

We are going to use nested v-for directs to loop through the students array and display the values of each student.

\temporary-student-ui.html – add the code in violet

<body>  
  
  
<!-- This div is rendered by Vuejs - Allows for Vuejs directives and componets -->  
<div class="container-fluid" id="app">  
 <b-jumbotron header="Temporary Student UI" lead="Quick and Dirty">  
 </b-jumbotron>

<table class="table table-hover table-striped">

<tr v-for="stu in students" :key="stu.id">  
 <td v-for="field in stu" >{{field}}</td>  
 </tr>  
</table>

<!— OPTIONAL: used to DEBUG the data properties/variables by rendering them on the page-->  
 <b-alert variant="secondary" dismissible show>  
 <pre>  
DATA:  
{{$data}}  
 </pre>  
 </b-alert>

</div>

<!— IMPORT: JS Libraries from the internet-->  
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>  
<script src="https://cdn.jsdelivr.net/npm/portal-vue/dist/portal-vue.umd.js"></script>  
<script src="https://cdn.jsdelivr.net/npm/bootstrap-vue/dist/bootstrap-vue.js"></script>  
<script>  
var vm = new Vue({  
 el: '#app', // css selector to find the div with id="app"  
 data:{ // essentially the app state - all the data currently used by the app for rendering  
 students:[ // contains all the student objects used to display in a table  
 {id:1, givenName:'Bob', familyName:'Smith', email:'bob@smith.ca'},  
 {id:2, givenName:'Sally', familyName:'Jones', email:'sally@jones.ca'},  
 {id:3, givenName:'Ernesto', familyName:'Basoalto', email:'basoaltoe@saskpolytech.ca'},  
 ],

},  
 methods:{

},  
 computed:{ // this section acts like dynamic data properties/variables  
  
 },  
 mounted(){ // the hook for the part of the lifecycle when the app is started and mounts the app element

}  
 });  
</script>  
</body>  
</html>

Exercise: Launch the webpage in webstorm’s built in webserver? What do you see?

Exercise: What is the port number in the URL? How many digits?

# Make a Request to the API

We were using Postman to make requests to the API, but now we need to tell the web brower to make requests to the API use the built in JavaScript function ***fetch***. We will learn more about ***fetch*** later in the course. For now, know that ***fetch*** returns a Promise (like a contract to return a response later). The resulting response object also returns a Promise when parsing the json from the server response.

Another issue is the we are using an async function with double arrow notation – so we CAN NOT use the key word ***‘this’*** in the function

\temporary-student-ui.html – add the code in violet – CAREFUL where you add the code

var ***app*** = new Vue({  
 el: '#app', // css selector to find the div with id="app"  
 data:{ // essentially the app state - all the data currently used by the app for rendering  
 students:[ // contains all the student objects used to display in a table  
 {id:1, givenName:'Bob', familyName:'Smith', email:'bob@smith.ca'},  
 {id:2, givenName:'Sally', familyName:'Jones', email:'sally@jones.ca'},  
 {id:3, givenName:'Ernesto', familyName:'Basoalto', email:'basoaltoe@saskpolytech.ca'},  
 ],  
 },  
 methods:{  
 apiRequest: async(method='GET', paramPath='/',data={})=>{  
 const url = 'http://localhost:3004/students' + paramPath;  
 const fetchOptions = {  
 credentials: 'include', // allows api to set cookies in the browser  
 referrerPolicy: 'strict-origin-when-cross-origin',  
 headers: { // headers required by lo4serverapi project otherwise 406 error  
 'Accept': 'application/json',  
 'X-Requested-With':'XmlHttpRequest',  
 'Content-Type':'application/json; charset=utf-8'  
 }  
 };  
 // ensure valid/allowed request methods  
 method = method.toUpperCase();  
 fetchOptions.method = ['GET','POST','PUT','DELETE'].includes(method) ? method: 'GET';  
  
 // convert JS object to JSON string – GET request cannot have a body property  
 if(fetchOptions.method != 'GET')fetchOptions.body = ***JSON***.stringify(data);  
  
 const res = await fetch(url,fetchOptions); // this the browser's version of Postman  
 ***console***.log(res);//OPTIONAL: help debug if we have issues  
  
 // if status code of the response in not in the 200s  
 if(!res.ok){  
 const error = new ***Error***(res.statusText + ':' + res.status);  
 error.status = res.status;  
 error.statusText = res.statusText;  
 error.data = await res.json();  
 throw error;  
 }  
  
 return res.json(); // convert response body/data INTO JSON  
 },  
 getStudents() { // uses apiRequest function to get an array of students from lo4serverapi project  
 this.apiRequest('GET')  
 .then(data => {  
 this.students = data; // the data should be a JSON array of student Objects  
 });  
 },  
 },  
 computed:{ // this section acts like dynamic data properties/variables  
  
 },  
 mounted(){ // the hook for the part of the lifecycle when the app is started and mounts the app element  
 this.getStudents();  
 }  
 });

Now because the webstorm’s built-in webserver uses a 5-digit port we need to make a slight change to the index.ts file of the lo4serverapi project

\src\index.ts – add the code in violet

// cors options  
const corsOptions ={  
 origin: /localhost\:\d{4,5}$/i, // localhost any 4 digit port  
 credentials: true, // needed to set and return cookies  
 allowedHeaders: 'Origin,X-Requested-With,Content-Type,Accept,Authorization',  
 methods: 'GET,PUT,POST,DELETE',  
 maxAge: 43200, // 12 hours  
};  
// create express app  
const app = express();  
  
  
createConnection().then(async (connection) => {  
// setup express app here  
 app.use(bodyParser.json()); // enable body parser  
  
 // OPTIONAL: add some lag to mimic internet traffic  
 app.use(async (req: express.Request, res: express.Response, next: express.NextFunction ) => {  
 await new ***Promise***((resolve) => setTimeout(resolve, 1200));  
 next();  
 });

Exercise: Start the lo4serverapi project (caution use the start configuration or ‘npm start’ in the terminal) once the project starts THEN run the temporary-student-ui.html. What do you see when the page loads?

If noting is happening open the developer tools in the browser and go to the console. Vue.js will output errors in the console

# Bootstrap-Vue - <https://bootstrap-vue.org/docs>

One of the best features of Vue.js is the ability to create components (self-contained re-usable Vue Instances often with their own html template and css). We will be creating our own single file components later, but for now we will use a 3rd party library of components.

Bootstrap-vue is a library of vue components that incorporate bootstrap css classes in each component. Using the components free a developer from applying bootstrap class to the elements. Some components have additional features built-in.

To use the components in our webpage simply put the component name in between < >. Like new custom html tags. All boostrap components start with a ‘b-‘ and you can use any of the vue-js directions like a normal html element. The format is

<b-component prop1 :prop2="object.property2" v-model="data.property3" @event4="doThis4" > </b-component>

## Bootstrap-Vue components - <https://bootstrap-vue.org/docs/components>

Some boostrap-vue components include:

[b-alert](https://bootstrap-vue.org/docs/components/alert#comp-ref-b-alert) – creates hidable boostrap alerts

<div>

<b-alert show variant="primary">Primary Alert</b-alert>

<b-alert show variant="secondary">Secondary Alert</b-alert>

<b-alert show variant="success">Success Alert</b-alert>

<b-alert show variant="danger">Danger Alert</b-alert>

<b-alert show variant="warning">Warning Alert</b-alert>

<b-alert show variant="info">Info Alert</b-alert>

<b-alert show variant="light">Light Alert</b-alert>

<b-alert show variant="dark">Dark Alert</b-alert>

</div>

[b-jumbotron](https://bootstrap-vue.org/docs/components/jumbotron) - creates a div with the boostrap jumbotron css class. Includes other properties for the header (add a h1 tag in the div) and lead (adds p tag in the div)

<b-jumbotron header="BootstrapVue" lead="Bootstrap v4 Components for Vue.js 2">

<p>For more information visit website</p>

<b-button variant="primary" href="#">More Info</b-button>

</b-jumbotron>

[b-form-group](https://bootstrap-vue.org/docs/components/form-group#component-reference) – creates a div with boostrap form group css classes. Include add properties like invalid-feeback (add a div with the invalid-feedback css class) and label ( add a label tag in the form group)

[b-form-input](https://bootstrap-vue.org/docs/components/form-input#component-reference) – creates an input tag along with other properties and directive like debounce (adds a delay before updating the underlying v-model) and state (adds the is-invalid or is-valid css class to the form group)

<b-form-group label="Province:" invalid-feedback="Province is not valid">

<b-form-input v-model=="province" :state="hasErrors?false:null"></b-form-input>

</b-form-group>

# Font-Awesome -

There is a trend to use icons or glyphs instead of text to indicate actions a user can perform. The main advantage is that is reduces the real-estate buttons take on the page AND does not require translating to other languages

Font-Awesome a library of css classes that insert icons into elements the have over 1600 free icons to use. <https://fontawesome.com/v5.15/icons?d=gallery&p=2&m=free>

<i class="fas fa-camera"></i> <!-- this icon's 1) style prefix == fas and 2) icon name == camera -->

<i class="fas fa-camera"></i> <!-- using an <i> element to reference the icon -->

<span class="fas fa-camera"></span> <!-- using a <span> element to reference the icon -->

# Add New Student

Lets a simple form interface to the table we made earlier along with a method to POST the new student to the server API

\temporary-student-ui.html – add code in violet to the html table

<table v-else class="table table-hover table-striped">  
 <tr>  
 <th>ID</th>  
 <th>Given Name</th>  
 <th>Family Name</th>  
 <th>Email</th>  
 <th>Phone</th>  
 <th>Address</th>  
 <th>Password</th>  
 <th>Action</th>  
 </tr> <!--HEADER-->  
<!-- CREATE INPUT-->  
 <tr>  
 <td></td>  
 <td>  
 <b-form-group :invalid-feedback="violation.givenName">  
 <b-form-input :state="violation.givenName?false:null" v-model="studentToCreate.givenName"/>  
 </b-form-group>  
 </td>  
 <td>  
 <b-form-group :invalid-feedback="violation.familyName">  
 <b-form-input :state="violation.familyName?false:null" v-model="studentToCreate.familyName" />  
 </b-form-group>  
 </td>  
 <td>  
 <b-form-group :invalid-feedback="violation.email" >  
 <b-form-input :state="violation.email?false:null" v-model="studentToCreate.email" />  
 </b-form-group>  
 </td>  
 <td>  
 <b-form-group :invalid-feedback="violation.phone">  
 <b-form-input :state="violation.phone?false:null" v-model="studentToCreate.phone" />  
 </b-form-group>  
 </td>  
 <td>  
 <b-form-group :invalid-feedback="violation.address" >  
 <b-form-input :state="violation.address?false:null" v-model="studentToCreate.address" />  
 </b-form-group>  
 </td>  
 <td>  
 <b-form-group :invalid-feedback="violation.password" >  
 <b-form-input :state="violation.password?false:null" v-model="studentToCreate.password" />  
 </b-form-group>  
 </td>  
 <td><button class="btn btn-primary far fa-save fa-2x" title="Create" @click="postStudent"/></td> <!--ACTIONS COLUMN-->  
 </tr>  
 <tr v-for="stu in students" :key="stu.id">  
 <td v-for="field in stu" >{{field}}</td>  
 </tr>  
 </table>

\temporary-student-ui.html – add code in violet to the vue instance

var ***app*** = new Vue({  
 el: '#app', // css selector to find the div with id="app"  
 data:{ // essentially the app state - all the data currently used by the app for rendering  
 violation:{}, // contains the student props that have error messages  
 studentToCreate:{}, //js object that stores the student data we are going to POST to the api  
 students:[ // contains all the student objects used to display in a table  
 {id:1, givenName:'Bob', familyName:'Smith', email:'bob@smith.ca'},  
 {id:2, givenName:'Sally', familyName:'Jones', email:'sally@jones.ca'},  
 {id:3, givenName:'Ernesto', familyName:'Basoalto', email:'basoaltoe@saskpolytech.ca'},  
 ],  
 },  
 methods:{  
 apiRequest: async(method='GET', paramPath='/',data={})=>{... collapsed code},   
 getStudents() { ... collapsed code},  
 postStudent() {  
 this.violation = {}; //clear errors from previous attempt  
 this.apiRequest('POST','/',this.studentToCreate)  
 .then(data=>{  
 // add the student to the students array so that the new student will appear in the table  
 this.students.push(data);  
 })  
 .catch(err=>{  
 if(err.status == 422) { // constraint violation object should have been returned by the server  
 // read the violations from the server and act accordingly  
 const temp = {};  
 err.data.forEach(vio=>{  
 ***Object***.assign(temp, {[vio.property] : vio.constraints[***Object***.keys(vio.constraints)[0]]});  
 })  
 this.violation = temp;  
 }  
 });  
 },  
 },  
 computed:{ // this section acts like dynamic data properties/variables  
  
 },  
 mounted(){ // the hook for the part of the lifecycle when the app is started and mounts the app element  
 this.getStudents();  
 }  
 });

# Add Loading Spinner and Search

We should always strive to improve the suer experience of our applications so with that in mind lets make a loading spinner visible while we are waiting for the server to respond to the GET students request. We will also add a search input that that will make use of the search functionality we added as part one of our previous exercises

\temporary-student-ui.html – add code in violet to the html table

<!-- This div is rendered by Vuejs - Allows for Vuejs directives and components -->  
<div class="container-fluid" id="app">  
 <b-jumbotron header="Temporary Student UI" lead="Quick and Dirty">  
 <b-input-group size="lg">  
 <b-form-input type="search" placeholder="Search" debounce="500" v-model="searchString" @update="getStudents"></b-form-input>  
 <b-input-group-append>  
 <b-button variant="info" class="fas fa-search fa-xl" @click="getStudents" ></b-button>  
 </b-input-group-append>  
 </b-input-group>  
 </b-jumbotron>  
  
 <div v-if="isLoading" class="text-center">  
 <b-spinner variant="primary" label="Spinning" style="width: 10rem; height:10rem;" />  
 </div>  
  
 <table v-else class="table table-hover table-striped">

.

.

.

</div>

\temporary-student-ui.html – add code in violet to the vue instance

var ***app*** = new Vue({  
 el: '#app', // css selector to find the div with id="app"  
 data:{ // essentially the app state - all the data currently used by the app for rendering  
 searchString:'', // bound to the search input and sent along with the readStudents method  
 violation:{}, // contains the student props that have error messages  
 studentToCreate:{}, //js object that stores the student data we are going to POST to the api  
 isLoading: false, // boolean to track if we are waiting for api request to return from server  
 students:[ // contains all the student objects used to display in a table  
 {id:1, givenName:'Bob', familyName:'Smith', email:'bob@smith.ca'},  
 {id:2, givenName:'Sally', familyName:'Jones', email:'sally@jones.ca'},  
 {id:3, givenName:'Ernesto', familyName:'Basoalto', email:'basoaltoe@saskpolytech.ca'},  
 ],  
 },  
 methods:{  
 apiRequest: async(method='GET', paramPath='/',data={})=>{. . . collapsed code },  
 getStudents() { // uses apiRequest function to get an array of students from lo4serverapi project  
 this.isLoading = true;  
 const paramPath = this.searchString.length? '/?search='+ this.searchString : '/';  
 this.apiRequest('GET',paramPath)  
 .then(data => {  
 this.students = data; // the data should be a JSON array of student Objects  
 })  
 .finally( ()=>{ this.isLoading = false; } );  
 },  
 postStudent() { . . . collapsed code },  
 },  
 computed:{ // this section acts like dynamic data properties/variables  
  
 },  
 mounted(){ // the hook for the part of the lifecycle when the app is started and mounts the app element  
 this.getStudents();  
 }  
 });

Exercise: Enter text in the search input. Notice how the searchString data property updates a half second AFTER you stop typing? This is the debounce feature.

Notice that event hook @update – this custom event created as part the b-form-input component. It is not a native JavaScript event   
Learn More: <https://bootstrap-vue.org/docs/components/form-input#comp-ref-b-form-input-events>

# Delete Student - Exercise

Lets a simple a button to each of the students and use an event handler to send a DELETE request to the server api. This time we send a parameter in the method call

\temporary-student-ui.html – add code in violet to the html table

<table v-else class="table table-hover table-striped">  
 <tr>  
 <th>ID</th>  
 <th>Given Name</th>  
 <th>Family Name</th>  
 <th>Email</th>  
 <th>Phone</th>  
 <th>Address</th>  
 <th>Password</th>  
 <th>Action</th>  
 </tr> <!--HEADER-->  
<!-- CREATE INPUT-->  
 <tr . . . collapsed code >

<tr v-for="stu in students" :key="stu.id">  
 <td v-for="field in stu" >{{field}}</td>

<td></td> <!--EMPTY PASSWORD-->  
 <td><button class="btn btn-danger far fa-trash-alt fa-2x" title="Delete" @click="deleteStudent(stu.id)"/></td>  
 </tr>  
 </table>

\temporary-student-ui.html – add code in violet to the vue instance

var ***app*** = new Vue({  
 el: '#app', // css selector to find the div with id="app"  
 data:{ // essentially the app state - all the data currently used by the app for rendering  
 violation:{}, // contains the student props that have error messages  
 studentToCreate:{}, //js object that stores the student data we are going to POST to the api  
 students:[ // contains all the student objects used to display in a table  
 {id:1, givenName:'Bob', familyName:'Smith', email:'bob@smith.ca'},  
 {id:2, givenName:'Sally', familyName:'Jones', email:'sally@jones.ca'},  
 {id:3, givenName:'Ernesto', familyName:'Basoalto', email:'basoaltoe@saskpolytech.ca'},  
 ],  
 },  
 methods:{  
 apiRequest: async(method='GET', paramPath='/',data={})=>{... collapsed code},   
 getStudents() { ... collapsed code},  
 postStudent() { ... collapsed code},

deleteStudent(id){  
 //EXERCISE: Make a delete request using apiRequest  
 // THEN Reload the list students to get the latest changes  
 alert(id); // remove this code on done coding the rest of delete  
 }  
 },

},  
 computed:{ // this section acts like dynamic data properties/variables  
  
 },  
 mounted(){ // the hook for the part of the lifecycle when the app is started and mounts the app element  
 this.getStudents();  
 }  
 });