



Avinash Nethala

Follow

Jun 17, 2018 · 8 min read

*Hello artisans, welcome to justlaravel.com. Here am going to show you how to perform CRUD operations with Vue.js and Laravel. I have previously discussed [Vue.js in my previous tutorials](#) where I created a [simple search app](#), and some [introduction to Vue](#), so here I am going to make a complete CRUD app using Vue.js and Laravel.*

Let's get started!



Vue.js CRUD App—justlaravel.com

## Setting up Vue js

We first need to setup Vue in our project. Its a breeze in Laravel as it is already included with Vue, we just need to install some modules to get started with Vue.

- Create a laravel project: `laravel new VueCRUD`
- Install PHP packages: go to VueCRUD directory and run `composer install`
- Generate a key: `php artisan key:generate`
- Install node packages: `npm install`
- Run the app: `npm run watch` to make the changes to take effect immediately and also run `php artisan serve` for the checking the actual app.

## Overview of the app









The app will display a list of people with the name, age and their profession, with edit and delete options, and 3 fields to add a new item.

**Name:**

**Age:**

**Profession:**

**+ ADD**

ID	Name	Age	Profession	Actions
27	Bob	24	Web Developer	 
28	Mary	22	Data Analyst	 
29	Siva	28	Bank Manager	 
30	Mani	26	Sales Representative	 

Vue.js CRUD App—justlaravel.com

## Initialize Vue js

All the Vue(js) part is written in file `app.js` in `/resources/assets/js` directory.

First I will create a new vue instance, and all the vue part is handled in an element with id `vue-crud-wrapper` i.e all the HTML content is wrapped in a div with id `vue-crud-wrapper` .

```
<body>
  ....
  <div id="vue-crud-wrapper">
    ...
    ...
    ...
  </div>
  ....
</body>
```

Create a new vue instance(`app.js`)

```
const app = new Vue({
  el: "#vue-crud-wrapper",
  data: { .... },
  methods: { .... },
  ....
});
```

## Add Person Form(Create Operation)

Here a form is shown with 3 fields name, age, and profession with a button which when clicked calls the action in the js and performs the add operation.

```

<div class="form-group">
  <label for="name">Name:</label>
  <input type="text" class="form-control" id="name"
name="name"
    required v-model="newItem.name" placeholder=" Enter
some name">
</div>
<div class="form-group">
  <label for="age">Age:</label>
  <input type="number" class="form-control" id="age"
name="age"
    required v-model="newItem.age" placeholder=" Enter
your age">
</div>
<div class="form-group">
  <label for="profession">Profession:</label>
  <input type="text" class="form-control" id="profession"
name="profession"
    required v-model="newItem.profession" placeholder="
Enter your profession">
</div>

<button class="btn btn-primary"
@click.prevent="createItem()" id="name" name="name">
<span class="glyphicon glyphicon-plus"></span> ADD
</button>

```

In the above snippet, for button, I used

`@click.prevent="createItem()"` which calls `createItem` vue method which is declared in the file `/resources/assets/js/app.js` .

So in the vue, I initialize some variables, which can be used in the methods like `createItem()`

```

var app = new Vue({
  el: '#vue-crud-wrapper',

  data: {
    ...
    items: [],
    hasError: true,
    newItem: { 'name': '', 'age': '', 'profession': '' },
  },
  ...
  ...

```

Here I also check for empty fields and throw an error when the input field is empty, so I initialize that `hasError` variable.

```

createItem: function createItem() {
  var _this = this;
  var input = this.newItem;

  if (input['name'] == '' || input['age'] == '' ||
input['profession'] == '' ) {
    this.hasError = false;
  } else {
    this.hasError = true;
    axios.post('/vueitems', input).then(function

```

```
(response) {
    _this.newItem = { 'name': '' };
    _this.getVueItems();
});
}
}
```

So when the fields are empty, I change the `hasError` variable to false, and a condition is written in the view to display error when the variable is set to false.

```
<p class="text-center alert alert-danger"
  v-bind:class="{ hidden: hasError }">Please fill all
  fields!</p>
```

Here the class `hidden` is bonded with the variable `hasError`, so if `hasError` is `true` that class is applied.

That is why I set `hasError` to `false` when there is an error.

Next, if all the fields are filled, I make a post call using `axios` and pass the data entered in the form. All the data that to be stored in the input variable and it is passed as an argument to the post function.

```
axios.post('/vueitems', input)
```

So this route `/vueitems` will call to a controller where the data is stored in the database.

In the `web.php` file at `/routes` directory

```
Route::post ( '/vueitems', 'MainController@storeItem' );
```

We can create a new controller by running the command `php artisan:make controller MainController`

Now in the `MainController(/app/Http/Controllers)`,

```
public function storeItem(Request $request) {
    $data = new Data ();
    $data->name = $request->name;
    $data->age = $request->age;
    $data->profession = $request->profession;
    $data->save ();
    return $data;
}
```

Here I used a model `Data` which has all the details related to database table the data is being stored.

So in the `app` directory, I create a new model named `Data.php` with the following class.

```
class Data extends Model {
    protected $table = "vueCrudData";
    public $timestamps = false;
}
```

Now when `$data->save();` is executed, all the data will be stored in the `vueCrudData` table in the database.

## Read Operation

Here the data is displayed in the table as shown in the image above. I use some vue properties and binding to the table elements so the data can be displayed as I wanted.

```
<div class="table table-borderless" id="table">
  <table class="table table-borderless" id="table">
    <thead>
      <tr>
        <th>ID</th>
        <th>Name</th>
        <th>Age</th>
        <th>Profession</th>
        <th>Actions</th>
      </tr>
    </thead>
    <tr v-for="item in items">
      <td>@{{ item.id }}</td>
      <td>@{{ item.name }}</td>
      <td>@{{ item.age }}</td>
      <td>@{{ item.profession }}</td>

      <td id="show-modal" @click="showModal=true;
setVal(item.id, item.name, item.age, item.profession)"
class="btn btn-info" ><span
      class="glyphicon glyphicon-pencil"></span></td>
      <td @click.prevent="deleteItem(item)" class="btn
btn-danger"><span
      class="glyphicon glyphicon-trash"></span>
    </td>
    </tr>
  </table>
</div>
```

In the above snippet, I used `v-for` directive to repeat all the `items` , also used some click methods which are used to update and delete items, which I will discuss in next section.

So the data in `items` variable is stored through js, let's see it.

There is a hook in vue called `mounted` . this hook calls immediately when a page loads or vue instance loads. So here in this hook, I call a method called `getVueItems()`, in this method a call to Laravel Controller and there it fetches the data and returns the same to view.

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

`getVueItems` method,

```
getVueItems: function getVueItems() {  
    var _this = this;  
  
    axios.get('/vueitems').then(function (response) {  
        _this.items = response.data;  
    });  
}
```

Now the data is set in the variable `items` , so iterating(v-for) this variable shows us all the data we need for the view.

So the route `/vueitems` routes to,

```
Route::get ( '/vueitems', 'MainController@readItems' );
```

`readItems` function in MainController,

```
public function readItems() {  
    $data = Data::all ();  
    return $data;  
}
```

The above snippet is very simple and straightforward, it gets all the data from the database and returns it and that data is shown in the view using Vue js.

## Update Operation

The table containing all the data is presented with two buttons “Edit” and “Delete” under “Actions” column.

So when edit button is clicked, a modal is shown with that particular row’s data and an option to edit them.

```
<td id="show-modal" @click="showModal=true; ..... </td>
```

the part, `@click="showModal= true;` triggers the modal,

```
<modal v-if="showModal" @close="showModal=false">
....
....
</modal>
```

here the `v-if` directive is responsible for toggling the modal open and close. So if the `showModal` variable is true modal is displayed else it is closed.

```
<td id="show-modal" @click="showModal=true; setVal(item.id,
item.name, item.age, item.profession)"
      class="btn btn-info"> <span class="glyphicon
glyphicon-pencil"></span> </td>
```

After setting `showModal` to `true`, a method `setVal` with id, name, age, and profession is set. So this function is used to display the data in the modal, the function's arguments are taken and assigned to other variables, which can be accessed in the view.

```
setVal(val_id, val_name, val_age, val_profession) {
  this.e_id = val_id;
  this.e_name = val_name;
  this.e_age = val_age;
  this.e_profession = val_profession;
}
```

Here the function's arguments `val_id`, `val_name`, `val_age`, `val_profession` are assigned to variables `e_id`, `e_name`, `e_age`, `e_profession` respectively.

So now these variables are used in the modal's body element.

```
<modal v-if="showModal" @close="showModal=false">
  <h3 slot="header">Edit Item</h3>
  <div slot="body">

    <input type="hidden" disabled class="form-control"
id="e_id" name="id"
      required :value="this.e_id">
    Name: <input type="text" class="form-control"
id="e_name" name="name"
      required :value="this.e_name">
    Age: <input type="number" class="form-control"
```

```

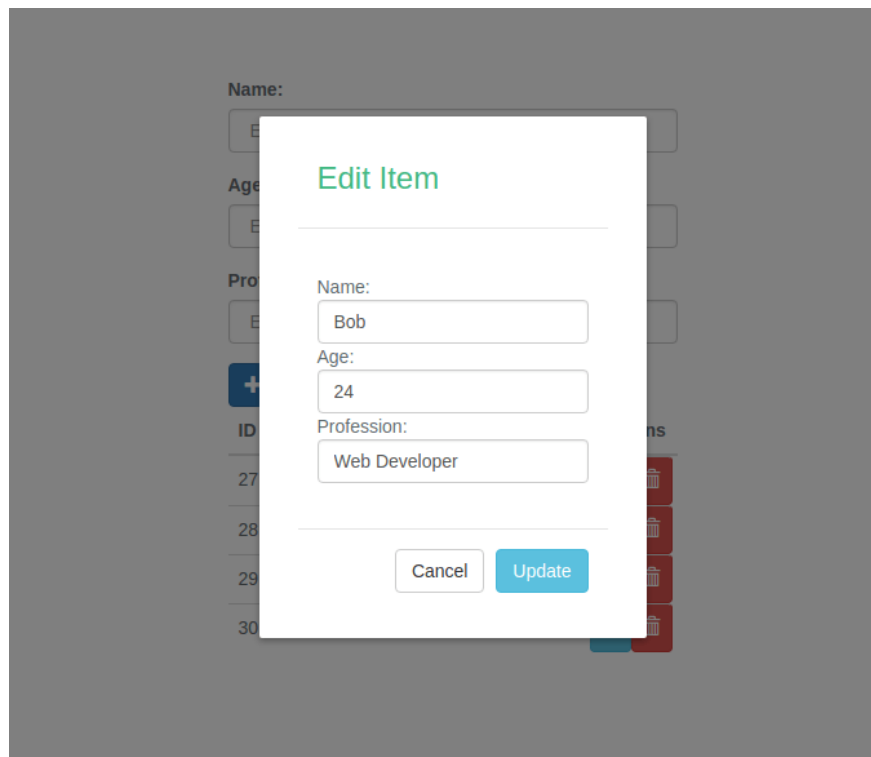
id="e_age" name="age"
    required :value="this.e_age">
    Profession: <input type="text" class="form-control"
id="e_profession" name="profession"
    required :value="this.e_profession">

</div>
<div slot="footer">
    <button class="btn btn-default" @click="showModal =
false">
        Cancel
    </button>

    <button class="btn btn-info" @click="editItem()">
        Update
    </button>
</div>
</modal>

```

If you see the input element's value `:value="this.e_age"` is the value that is set in the js script above in `setVal` method.



Vue js CRUD App—justlaravel.com

In the modal footer section, two buttons, “Cancel” and “Update” are present. So when cancel is clicked, the `showModal` variable is set to `false` so the modal closes, when Update is clicked a method `editItem()` is called.

`editItem()` vue js method

```

editItem: function(){
    var i_val = document.getElementById('e_id');
    var n_val = document.getElementById('e_name');
    var a_val = document.getElementById('e_age');
    var p_val = document.getElementById('e_profession');

```



```

        axios.post('/edititems/' + i_val.value, {val_1:
n_val.value, val_2: a_val.value,val_3: p_val.value })
        .then(response => {
            this.getVueItems();
            this.showModal=false
        });
    }

```

The first four lines gets all the info from the modal using

`getElementById` and uses those to pass as a an arguments to axios post method.

So in response method, `getVueItems()` is again called so the items are updated in the view.

And the modal is closed by setting `showModal` to `false` .

The `edititems/id` route is set to,

```

Route::post ( '/edititems/{id}', 'MainController@editItem'
);

```

The editItem function in MainController,

```

public function editItem(Request $request, $id){
    $data =Data::where('id', $id)->first();
    $data->name = $request->get('val_1');
    $data->age = $request->get('val_2');
    $data->profession = $request->get('val_3');
    $data->save();
    return $data;
}

```

The above snippet will fetch the row with the given id, and update its elements name, age, and profession.

## Delete Operation

In the actions column there is an icon for delete,

```

<td @click.prevent="deleteItem(item)" class="btn btn-
danger"><span
        class="glyphicon glyphicon-trash"></span></td>

```

So when clicked it calls `deleteItem(item)` method.

```
deleteItem: function deleteItem(item) {
    var _this = this;
    axios.post('/vueitems/' + item.id).then(function
(response) {
    _this.getVueItems();
    _this.hasDeleted = false
    });
}
```

this method simply calls `/vueitems/id` route which calls Laravel `deleteItem` function.

the route,

```
Route::post ( '/vueitems/{id}', 'MainController@deleteItem'
);
```

`deleteItem` function in MainController,

```
public function deleteItem(Request $request) {
    $data = Data::find ( $request->id )->delete ();
}
```

he item gets deleted, and in the vue response function `getVueItems` is called, do the items are updated.







**Name:**

**Age:**

**Profession:**

**+ ADD**

Deleted Successfully!

ID	Name	Age	Profession	Actions
28	Mary	22	Data Analyst	 
29	Siva	28	Bank Manager	 
30	Mani	26	Sales Representative	 

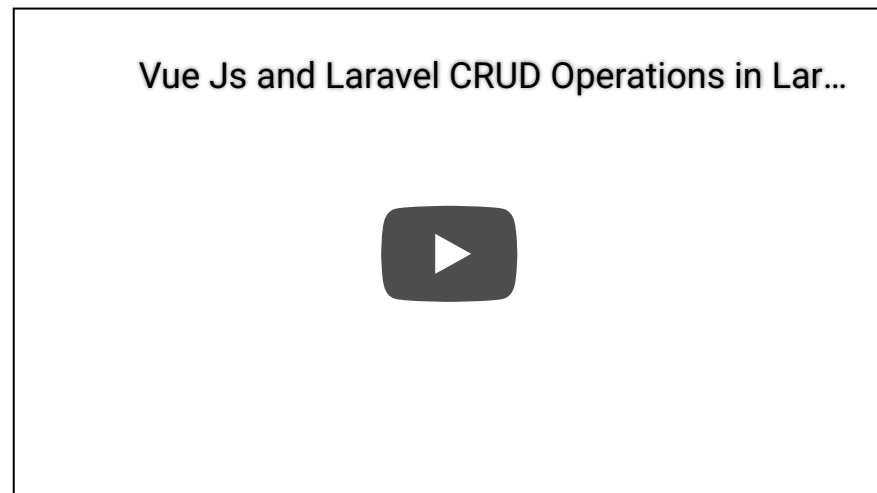
So finally all the CRUD(Create, Read, Update, Delete) operations are performed using vue js and laravel.

. . .

Post: <https://justlaravel.com/vue-js-crud-laravel/>

<https://justlaravel.com/vue-js-crud-laravel/>

YouTube: <https://www.youtube.com/watch?v=HeDo8v0TNDw>



Working Demo: <http://demos.justlaravel.com/vue-js-crud-laravel>

<p>Vue.js CRUD - Just Laravel</p> <p>This is a live demo of the application performing CRUD Operations in Laravel using Vue.js. Vue Js... demos.justlaravel.com</p>	
---	--

GitHub: <https://github.com/avinashn/vue-js-crud-laravel>

<p>avinashn/vue-js-crud-laravel</p> <p>vue-js-crud-laravel - Simple CRUD operations using Laravel and Vue.js</p> <p>github.com</p>	A square profile picture of a man with dark hair, a beard, and glasses, wearing a dark shirt. He is standing in front of a Christmas tree decorated with lights and ornaments.
--	--



