# Week 05 - Front-end

This material can be found in this [link](https://codeberg.org/kaduardo/shu-aaf/src/branch/main/week08-front-end/).

## 1. Introduction

This week you will build a front-end for a Web application. We’ll start by modifying the existing petshop application to remove pug and make sure our endpoints are fully RESTful and return JSON objects.

You will then move on to building an implementation of a frontend for the petshop application exploring the Vue framework.

## 2. Refactoring the back-end

We will begin by restructuring your express/pug project so that it is accessible from other applications and so that it return JSON rather than rendered HTML pages. At this point we are removing the requirement for pug.

Make a copy of you petshop project and open it in your IDE/editor. In these instructions we are assuming that you have been following the naming convetions from previous exercises and the name of your backend folder is petshop/backend.

\*You can use one of the sample solutions from past exercise with NodeJS+Express+Mongoose as starting point (It is saved as [petshop-with-pug](solutions/petshop-with-pug) inside the solutions directory.

* Make sure to create a copy of this directory so you can go back to it in case of any mistakes.
* Make sure to run npm install before starting the application.

### 2.1. Configuration

We will start by configuring the backend project so that it runs on a different port. In the bin folder you will find a file called www. Look for the following line and change the port to be **3050**:

var port = normalizePort(process.env.PORT || '3000');

You can test your application by accessing <http://localhost:3050/petshop>.

*Don’t forget to start you mongodb database.*

### 2.2. Returning JSON

Now we want to change the project so that it returns JSON documents to calling clients rather than rendering its results as HTML. The only changes that you need to make are in the file that holds the controller (controllers/animal.controller.js).

Essentially, you have to comment out the lines that return data either directly or by rendering the template, and then replace them with code that return json objects

Start with the findAll method (which implements the GET route). In this case, you comment out the code that render the pug page with the data returned from the database:

//...  
 res.render('animals',  
 {title: 'Pet Store',  
 animals: data});  
//...

And replace it with:

return res.send(data);

For your reference, this is one example of the findAll method after the modifications. Remember this is here just as a reference as you should be working on your own code from the previous weeks.

// Retrieve all Animals from the database.  
exports.findAll = (req, res) => {  
 const name = req.query.name;  
 var condition = name ? { name: { $regex: new RegExp(name), $options: "i" } } : {};  
 Animal  
 .find(condition)  
 .then(data => {  
 //res.render('animals',  
 // {title: 'Pet Store',  
 // animals: data});  
 res.send(data);  
 })  
 .catch(err => {  
 res.status(500).send({  
 message:  
 err.message || "Some error occurred while retrieving Animals."  
 });  
 });  
 };

Now working with the create method, which implements the POST route.

* Comment out the redirect call

//...  
res.redirect('/petshop/animals');  
//...

and replace it with

res.send(data);

For your reference this is one example of the create method after the changes.

// Create and Save a new Animal  
exports.create = (req, res) => {  
 // Validate request  
 if (!req.body.name) {  
 res.status(400).send({ message: "Content can not be empty!" });  
 return;  
 }  
   
 // Create a Animal model object  
 const animal = new Animal({  
 name: req.body.name,  
 species: req.body.species,  
 breed: req.body.breed,  
 age: req.body.age,  
 colour: req.body.colour  
 });  
   
 // Save Animal in the database  
 animal  
 .save()  
 .then(data => {  
 console.log("Animal saved in the database: " + data);  
 //res.redirect('/petshop/animals');  
 res.send(data);  
 })  
 .catch(err => {  
 res.status(500).send({  
 message:  
 err.message || "Some error occurred while creating the Animal."  
 });  
 });  
 };

Now you need to test your backend. Access the GET route (don’t forget the change of port) from the browser and you should get some JSON back.

From this point on you have the choice of implementing a front-end using [Vue](#X4ad1007e8f44c165e1b5c100035a6156a380d57) or [React](#X3bcebb586a5efe066c66a45dfc7d025cf7368f3). These instructions consider both scenarios.

## 3. Creating a Vue front-end

In this part of the lab you will build a front-end for the petshop Web application using Vue. You will begin by building simple components that demonstrate how the framework is used and how you can connect your frontend to the express petshop application from previous exercises. Use the documentation at <https://vuejs.org/guide/introduction.html> to help you with this session.

### 3.1. Set-up and initialisation

You must make sure mongodb server and the express backend of the petshop application are running before starting. The first step is to install the Vue CLI by running the command shown below:

npm install -g @vue/cli

This command installs Vue CLI globally and it can now be accessed from the terminal. Confirm Vue CLI installation by running:

vue --version

You can now create your Vue application. Open a command prompt or terminal and change to a new directory where you want to store your vue files. This should not be the same place as your express work. We are considering that your directory is named frontend-vue inside petshop.

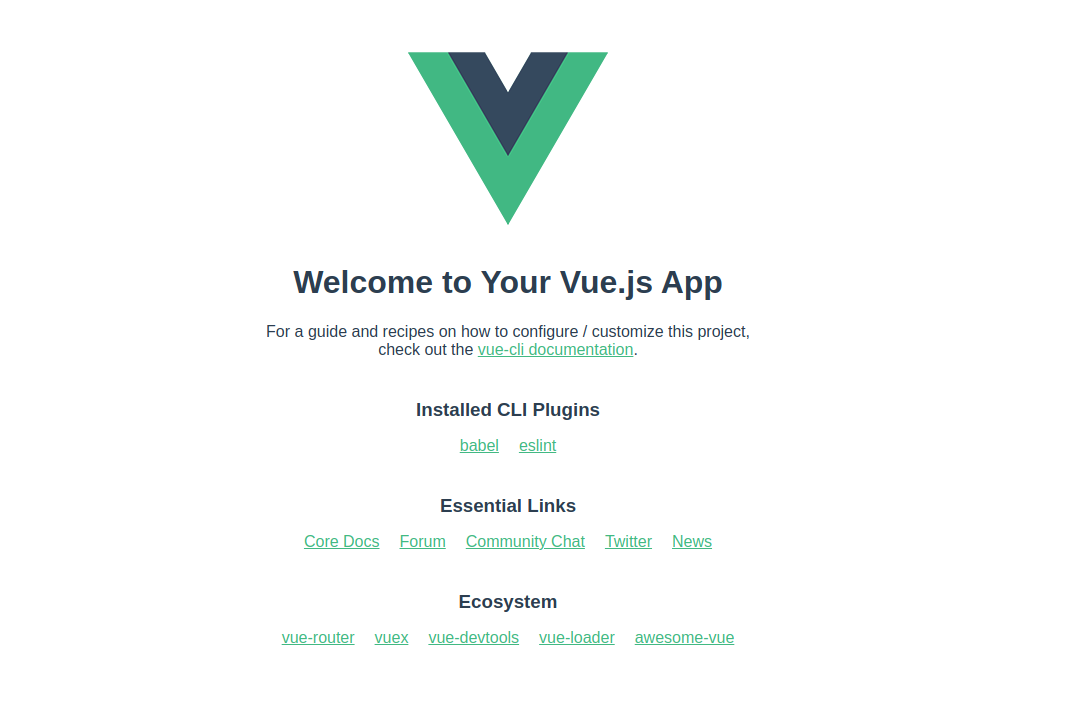
Type the following command:

vue create frontend-vue

You will be prompted to pick a preset. For now the default preset for Vue 3 is fine. Choose NPM as package manager in the next question. After vue CLI finishes creating your project you can see how it looks like by starting the application an opening it in your web browser

cd frontend-vue  
npm run serve

Open your browser at <http://localhost:8080/>. You should see something that looks like the image below.



First screen after loading your Vue application.

### 3.2. Adding Bootstrap

We are going to use bootstrap in our front-end. Run the following command in your terminal to add it to your project:

npm install bootstrap jquery popper.js

Now add it to your src/main.js file.

import { createApp } from 'vue'  
import App from './App.vue'  
import 'bootstrap'  
import 'bootstrap/dist/css/bootstrap.min.css'  
...

### 3.3. Creating a Data Service

Let’s create a data service that will act as a helper module for communicating with the backend sending http REST requests to the back-end. We’ll use axios for handling this communication. You can install it with in your front-end directory:

npm install axios

Create a new file under src called http-common.js with the following content:

import axios from "axios";  
  
export default axios.create({  
 baseURL: "http://localhost:3050/petshop",  
 headers: {  
 "Content-type": "application/json"  
 }  
});

Create a new folder called services and then a new file called AnimalDataService.js inside this folder. Use the following code for this file:

import http from "../http-common";  
  
class AnimalDataService {  
  
 getAll() {  
 return http.get("/animals");  
 }  
  
 get(id) {  
 return http.get(`/animals/${id}`)  
 }  
  
 create(data) {  
 return http.post("/animals", data);  
 }  
  
 update(id, data) {  
 return http.put(`/animals/${id}`, data);  
 }  
  
 delete(id) {  
 return http.delete(`/animals/${id}`);  
 }  
  
 deleteAll() {  
 return http.delete(`/animals`);  
 }  
  
 findByName(name) {  
 return http.get(`/animals?name=${name}`);  
 }   
}  
  
export default new AnimalDataService();

### 3.4. Listing animals component

We will begin by creating Vue components to list our animals. Inside the existing folder called components create a file named AnimalsList.vue. Use the following code for this components:

<template>  
 <div class="list row">  
 <div class="col-md-8">  
 <div class="input-group mb-3">  
 <input type="text" class="form-control" placeholder="Search by name"  
 v-model="name"/>  
 <div class="input-group-append">  
 <button class="btn btn-outline-secondary" type="button"  
 @click="searchName"  
 >  
 Search  
 </button>  
 </div>  
 </div>  
 </div>  
 <div class="col-md-6">  
 <h4>Animals List</h4>  
 <ul class="list-group">  
 <li class="list-group-item"  
 :class="{ active: index == currentIndex }"  
 v-for="(animal, index) in animals"  
 :key="index"  
 @click="setActiveAnimal(animal, index)"  
 >  
 {{ animal.name }}  
 </li>  
 </ul>  
  
 <button class="m-3 btn btn-sm btn-danger" @click="removeAllAnimals">  
 Remove All  
 </button>  
 </div>  
 <div class="col-md-6">  
 <div v-if="currentAnimal">  
 <h4>Animal</h4>  
 <div>  
 <label><strong>Name:</strong></label> {{ currentAnimal.name }}  
 </div>  
 <div>  
 <label><strong>Species:</strong></label> {{ currentAnimal.species }}  
 </div>  
 <div>  
 <label><strong>Breed:</strong></label> {{ currentAnimal.breed }}  
 </div>  
 <div>  
 <label><strong>Age:</strong></label> {{ currentAnimal.age }}  
 </div>  
 <div>  
 <label><strong>Colour:</strong></label> {{ currentAnimal.colour }}  
 </div>  
  
 <router-link :to="'/animals/' + currentAnimal.\_id" class="badge badge-warning">Edit</router-link>  
 </div>  
 <div v-else>  
 <br />  
 <p>Please click on an Animal...</p>  
 </div>  
 </div>  
 </div>  
</template>  
  
<script>  
import AnimalDataService from "../services/AnimalDataService";  
  
export default {  
 name: "animals-list",  
 data() {  
 return {  
 animals: [],  
 currentAnimal: null,  
 currentIndex: -1,  
 name: ""  
 };  
 },  
 methods: {  
 retrieveAnimals() {  
 AnimalDataService.getAll()  
 .then(response => {  
 this.animals = response.data;  
 console.log(response.data);  
 })  
 .catch(e => {  
 console.log(e);  
 });  
 },  
  
 refreshList() {  
 this.retrieveAnimals();  
 this.currentAnimal = null;  
 this.currentIndex = -1;  
 },  
  
 setActiveAnimal(animal, index) {  
 this.currentAnimal = animal;  
 this.currentIndex = animal ? index : -1;  
 },  
  
 removeAllAnimals() {  
 AnimalDataService.deleteAll()  
 .then(response => {  
 console.log(response.data);  
 this.refreshList();  
 })  
 .catch(e => {  
 console.log(e);  
 });  
 },  
   
 searchName() {  
 AnimalDataService.findByName(this.name)  
 .then(response => {  
 this.animals = response.data;  
 this.setActiveAnimal(null);  
 console.log(response.data);  
 })  
 .catch(e => {  
 console.log(e);  
 });  
 }  
 },  
 mounted() {  
 this.retrieveAnimals();  
 }  
};  
</script>  
  
<style>  
.list {  
 text-align: left;  
 max-width: 750px;  
 margin: auto;  
}  
</style>

Notice the use of AnimalDataService and the call to getAll() method.

### 3.5. Creating a router for your component

Now let’s create a router.js file inside the src folder. Use the following code for your vue router:

import { createWebHistory, createRouter } from 'vue-router';  
   
const routes = [  
 {  
 path: "/",  
 alias: "/animals",  
 name: "animals",  
 component: () => import("./components/AnimalsList")  
 }  
];  
  
const router = createRouter({  
 history: createWebHistory(),  
 routes  
});  
  
export default router;

Notice how we only created route for the first component. We’ll come back to add new routes as we add more components.

You need to register your router in the main.js file. We will include an import call for the router and then include an use call in the application.

...  
import router from './router';  
  
createApp(App).use(router).mount('#app');

### 3.6. Creating the navbar with our new route

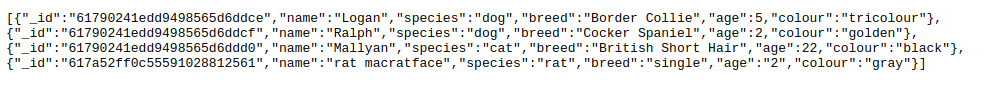
To conclude, update the app.vue to include the newly created route in your navigation bar.

<template>  
 <div id="app">  
 <nav class="navbar navbar-expand navbar-dark bg-dark">  
 <router-link to="/" class="navbar-brand">petshop</router-link>  
 <div class="navbar-nav mr-auto">  
 <li class="nav-item">  
 <router-link to="/animals" class="nav-link">Animals</router-link>  
 </li>  
  
 <li class="nav-item">  
 <router-link to="/add-animal" class="nav-link">Add</router-link>  
 </li>  
  
 </div>  
 </nav>  
  
 <div class="container mt-3">  
 <h2>Petshop Vue 3 example</h2>  
 <router-view />  
 </div>  
 </div>  
</template>  
  
<script>  
export default {  
 name: "app"  
};  
</script>

### 3.7. Running

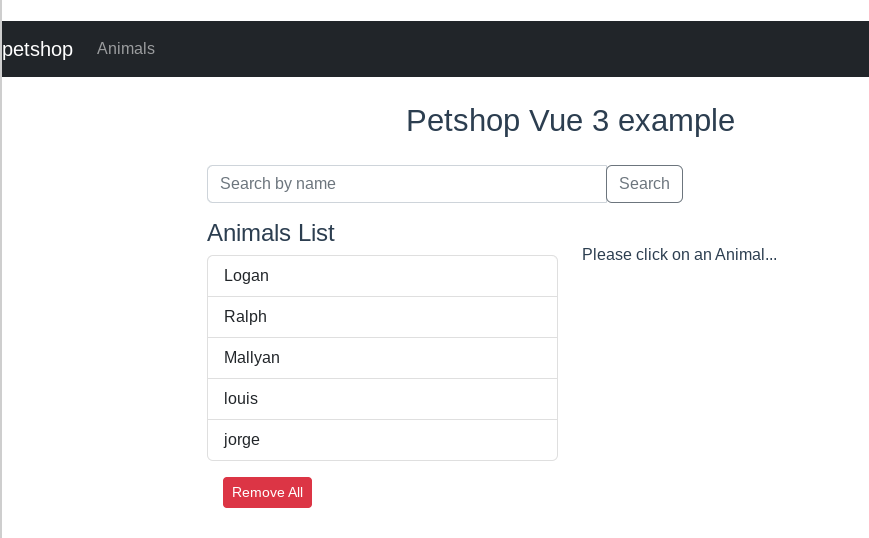
Make sure your database and back-end are running.

Now open your browser to <http://localhost:3050/petshop/animals> and you should see something similar to the screen below.



Example of response from the back-end applcation.

Open a new tab in your browser and go to http://localhost:8080/animals, you should see a screen similar to this one.



Example of screen using Vue framework.

If you have any problems use the console from your browser to identify any error messages. One of the most common errors is related to cors (Cross-Origin Request). If you’ve been following the lab session throughout the semester your back-end should be configure for cors and ready to go. If not, or if you created a new back-end project from scratch you need to go in the folder of your Express application and install the cors module with the following command:

npm install cors

And now update the app.js file to include the cors module by adding the following line in the beginning of the file.

var cors = require('cors');

Then configure your application to use cors by including the following line:

app.use(cors());

Once this is done, start your back-end.

### 3.8. Getting animal details component

This is a component to display information of an specific animal, allowing the user to edit and delete this animal.

Create a new component AnimalDetails.vue with the following code:

<template>  
 <div v-if="currentAnimal" class="edit-form">  
 <h4>Animal</h4>  
 <form>  
 <div class="form-group">  
 <label for="name">Name</label>  
 <input type="text" class="form-control" id="name"  
 v-model="currentAnimal.name"  
 />  
 </div>  
 <div class="form-group">  
 <label for="species">Species</label>  
 <input type="text" class="form-control" id="species"  
 v-model="currentAnimal.species"  
 />  
 </div>  
 <div class="form-group">  
 <label for="breed">Breed</label>  
 <input type="text" class="form-control" id="breed"  
 v-model="currentAnimal.breed"  
 />  
 </div>  
 <div class="form-group">  
 <label for="age">Age</label>  
 <input type="text" class="form-control" id="age"  
 v-model="currentAnimal.age"  
 />  
 </div>  
 <div class="form-group">  
 <label for="colour">Colour</label>  
 <input type="text" class="form-control" id="colour"  
 v-model="currentAnimal.colour"  
 />  
 </div>  
 </form>  
   
 <button class="badge badge-danger mr-2"  
 @click="deleteAnimal"  
 >  
 Delete  
 </button>  
   
 <button type="submit" class="badge badge-success"  
 @click="updateAnimal"  
 >  
 Update  
 </button>  
 <p>{{ message }}</p>  
 </div>  
   
 <div v-else>  
 <br />  
 <p>Please click on an Animal...</p>  
 </div>  
 </template>  
   
 <script>  
 import AnimalDataService from "../services/AnimalDataService";  
   
 export default {  
 name: "animal-details",  
 data() {  
 return {  
 currentAnimal: null,  
 message: ''  
 };  
 },  
 methods: {  
 getAnimal(id) {  
 AnimalDataService.get(id)  
 .then(response => {  
 this.currentAnimal = response.data;  
 console.log(response.data);  
 })  
 .catch(e => {  
 console.log(e);  
 });  
 },  
   
 updateAnimal() {  
 AnimalDataService.update(this.currentAnimal.\_id, this.currentAnimal)  
 .then(response => {  
 console.log(response.data);  
 this.message = 'The animal was updated successfully!';  
 })  
 .catch(e => {  
 console.log(e);  
 });  
 },  
   
 deleteAnimal() {  
 AnimalDataService.delete(this.currentAnimal.\_id)  
 .then(response => {  
 console.log(response.data);  
 this.$router.push({ name: "animals" });  
 })  
 .catch(e => {  
 console.log(e);  
 });  
 }  
 },  
 mounted() {  
 this.message = '';  
 this.getAnimal(this.$route.params.id);  
 }  
 };  
 </script>  
   
 <style>  
 .edit-form {  
 max-width: 300px;  
 margin: auto;  
 }  
 </style>

We need to update our routes with the path for the details page. Edit the router.js file to add the following code:

{  
 path: "/animals/:id",  
 name: "animal-details",  
 component: () => import("./components/AnimalDetails")  
 }

### 3.9. Adding new animals

Let’s now create a page to add new animals into our database. The first step is to create the Vue component that will hold the HTML form. Inside the src/components folder create a file named AnimalsAdd.vue and use the following as its content:

<template>  
 <div class="submit-form">  
 <div v-if="!submitted">  
 <div class="form-group">  
 <label for="name">Name</label>  
 <input  
 type="text"  
 class="form-control"  
 id="name"  
 required  
 v-model="animal.name"  
 name="name"  
 />  
 </div>  
  
 <div class="form-group">  
 <label for="species">Species</label>  
 <input  
 class="form-control"  
 id="species"  
 required  
 v-model="animal.species"  
 name="species"  
 />  
 </div>  
  
 <div class="form-group">  
 <label for="breed">Breed</label>  
 <input  
 class="form-control"  
 id="breed"  
 required  
 v-model="animal.breed"  
 name="breed"  
 />  
 </div>  
  
 <div class="form-group">  
 <label for="age">Age</label>  
 <input  
 class="form-control"  
 id="age"  
 required  
 v-model="animal.age"  
 name="age"  
 />  
 </div>  
  
 <div class="form-group">  
 <label for="colour">Colour or markings</label>  
 <input  
 class="form-control"  
 id="colour"  
 required  
 v-model="animal.colour"  
 name="colour"  
 />  
 </div>  
  
 <button @click="saveAnimal" class="btn btn-success">Submit</button>  
 </div>  
  
 <div v-else>  
 <h4>You submitted successfully!</h4>  
 <button class="btn btn-success" @click="newAnimal">Add</button>  
 </div>  
 </div>  
</template>  
  
<script>  
import AnimalDataService from "../services/AnimalDataService";  
  
export default {  
 name: "add-animal",  
 data() {  
 return {  
 animal: {  
 id: null,  
 name: "",  
 species: "",  
 breed: "",  
 age: "",  
 colour: ""  
 },  
 submitted: false  
 };  
 },  
 methods: {  
 saveAnimal() {  
 var data = {  
 name: this.animal.name,  
 species: this.animal.species  
 //Complete with the other fields  
 };  
  
 AnimalDataService.create(data)  
 .then(response => {  
 this.animal.id = response.data.id;  
 console.log(response.data);  
 this.submitted = true;  
 })  
 .catch(e => {  
 console.log(e);  
 });  
 },  
   
 newAnimal() {  
 this.submitted = false;  
 this.animal = {};  
 }  
 }  
};  
</script>  
  
<style>  
.submit-form {  
 max-width: 300px;  
 margin: auto;  
}  
</style>

Now let’s update our routes (router.js) by adding a new route to /add-animal.

...  
 {  
 path: "/add-animal",  
 name: "add-animal",  
 component: () => import("./components/AnimalAdd")  
 }  
 ...

Finally we update our navbar (inside App.vue) adding a new entry with a link to our new page:

...  
 <li class="nav-item">  
 <router-link to="/add-animal" class="nav-link">Add</router-link>  
 </li>  
 ...

Now everything ought to work. Load your Vue frontend in the browser to see it connected to the database. Remember:

* You need to have the database running in a terminal.
* You must be running the backend in a second terminal.
* Run the frontend project from a third terminal.

### 4. Exercises with Vue

1. Add some error handling to the application so that invalid or meaningless data (or incomplete animals) cannot be added to the system. You may want to do this using an HTML5 form (<https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/Form_validation>).
2. Implement the ability of visualising only one animal. You can do this by using the \_id field that MongoDB generates for each document.
3. Now implement the ability to edit each item. You can do this by using the \_id field that MongoDB generates for each document and a simple form.
4. Add error handling code for database calls.
5. Add additional code to return HTTP status errors if resources (including routes) are unavailable. Make sure your front end is considering those return codes and showing appropriate error messages when this is the case.
6. Add some style to the page. Look at Bootstrap (<https://getbootstrap.com/docs/3.3/css/>) or Google’s Material design (<https://material.io/>) for inspiration or code.
7. Modify the application so that each animal can be represented by a small avatar. Display the avatar as a circular image.

## 5. References

Some links that have been consulted in writing these instructions and other tutorials for you to practice with:

* BezKoder, Vue 3 CRUD example with Axios & Vue Router - <https://www.bezkoder.com/vue-3-crud/>. (This is the main reference for the Vue front-end.)
* BezKoder, Vue.js + Node.js + Express + MongoDB example: MEVN stack CRUD Application - <https://bezkoder.com/vue-node-express-mongodb-mevn-crud/>. (A single reference point for the full-stack)