Ing. Luis Guillermo Molero Suárez

Aplicación CRUD de Stack MEVN Vue

En este ejercicio, se creará una aplicación básica de una sola página del stack MEVN desde cero para estudiantes. Esta aplicación permitirá crear estudiantes, mostrar la lista de estudiantes, actualizarlos y eliminarlos. Finalmente, se creará un servidor usando Node Js y Express.js y almacenaremos los registros de los estudiantes, utilizando MongoDB y gestionando el front-end de la aplicación con Vue.js.

Configuración:

- Cree un nuevo proyecto de Vue
- Añadiendo Bootstrap en Vue
- Construir componentes de Vue
- Habilitar el enrutador Vue
- Configuración de la vista de navegación y enrutador
- Agregar Axios en Vue.js
- Construye formularios en Vue con Bootstrap 4
- Configuración del entorno del servidor de nodo
- Configurar la base de datos MongoDB
- Crear modelo de mongoose
- Crear ruta en la aplicación Node / Express
- Iniciar aplicación Node / Express
- Cree datos de estudiantes con Axios POST
- Mostrar lista de datos y eliminar datos en Vue
- Actualizar datos con solicitud POST

Prerequisitos

Antes de comenzar con este ejercicio, se debe conocer los fundamentos de Vue, HTML, CSS, JavaScript, TypeScript o ES6. Consulte el sitio web oficial de Vue para obtener más información sobre sus funciones, conceptos centrales y referencia.

https://vuejs.org/

Para instalar el proyecto Vue, debemos tener Vue CLI instalado en nuestro sistema de desarrollo. En caso contrario, abra un CMD y ejecute el siguiente comando:

```
npm install -g @vue/cli
```

Crear aplicación Vue

Comencemos a construir el proyecto Vue con “vue create”

```
vue create vue-mevn-stack-app
```

Ingrese a la carpeta del proyecto Vue:

```
cd vue-mevn-stack-app
```

Para iniciar el proyecto Vue MEVN, ejecute el siguiente comando:

```
npm run serve
```

Este comando abre el proyecto Vue en la siguiente URL:

http://localhost:8080/

Integración de Bootstrap con la aplicación Vue

En el siguiente paso, se instalará el framework Bootstrap en nuestra aplicación de Stack MEVN. Este framework nos permitirá usar el componente UI de Bootstrap en nuestra aplicación CRUD Vue.

```
npm install bootstrap
```

Importe la ruta del framework Bootstrap dentro del archivo main.js. Ahora, puede usar los componentes de la interfaz de usuario de Bootstrap en su app. https://vuejs.org/v2/api/

```
import Vue from "vue";
import App from "./App.vue";
import router from "./router";

import "bootstrap/dist/css/bootstrap.min.css";

Vue.config.productionTip = false;

new Vue({
  router,
  render: (h) => h(App),
}).$mount("#app");
```

Creación de componentes Vue

Dirígete al directorio “src/components”, aquí tenemos que crear los siguientes componentes. Estos componentes manejarán los datos en nuestra aplicación Vue.Js.

- CreateComponent.vue
- EditComponent.vue
- ListComponent.vue

Abra el archivo “src/components/CreateComponent.vue” y agregue el siguiente código dentro de él.

```
<template>
    <div class="row justify-content-center">
        <div class="col-md-6">
            <!-- Content goes here -->
        </div>
    </div>
</template>

<script>
    export default {
        data() {
            return {
            }
        }
    }
</script>
```

Abra el archivo “src/components/EditComponent.vue” y coloque el código dentro de él.

```
<template>
  <div class="row justify-content-center">
    <div class="col-md-6">
      <!-- Update Student content -->
    </div>
  </div>
</template>

<script>
export default {
  data() {
    return {};
  },
};
</script>
```

Abra el archivo “src/components/ListComponent.vue” y agregue el siguiente código en él.

```
<template>
  <div class="row justify-content-center">
    <div class="col-md-6">
      <!-- Display Student List -->
    </div>
  </div>
</template>

<script>
export default {
  data() {
    return {};
  },
};
</script>
```

Habilitar el enrutador Vue

Instale la dependencia vue-router

```
npm install --save vue-router
```

Instalar Router de vue

```
vue add router
```

Abra “src/router/index.js” y reemplace el código existente con el siguiente código.

```
import Vue from "vue";
import VueRouter from "vue-router";

Vue.use(VueRouter);

const routes = [
  {
    path: "/",
    name: "home",
    component: () => import("../components/CreateComponent"),
  },
  {
    path: "/view",
    name: "view",
    component: () => import("../components/ListComponent"),
  },
  {
    path: "/edit/:id",
    name: "edit",
```

```
    component: () => import("../components/EditComponent"),
  },
];

const router = new VueRouter({
  mode: "history",
  base: process.env.BASE_URL,
  routes,
});

export default router;
```

Configuración de la navegación con Bootstrap y Router View en Vue

Vaya al archivo “src/App.vue”, aquí definimos el componente Bootstrap Navigation, la directiva de vista de enrutador y la directiva de enlace de enrutador.

El <router-link> es el componente para facilitar la navegación del usuario en una aplicación vue habilitada para enrutadores.

El componente <router-view> es el componente principal responsable de representar el componente coincidente para la ruta proporcionada. https://router.vuejs.org/api/#event

```
<template>
  <div>
    <!-- Nav bar -->
    <nav
      class="navbar navbar-dark bg-primary justify-content-between flex-
nowrap flex-row"
    >
      <div class="container">
        <a class="navbar-brand float-left">MEVN Stack Example</a>
        <ul class="nav navbar-nav flex-row float-right">
          <li class="nav-item">
            <router-link class="nav-link pr-3" to="/"
              >Create Student</router-link
            >
          </li>
          <li class="nav-item">
            <router-link class="nav-link" to="/view">View Students</router-link>
          </li>
        </ul>
      </div>
    </nav>

    <!-- Router view -->
    <div class="container mt-5">
      <router-view></router-view>
    </div>
  </div>
</template>
```

Agregue Axios en Vue.js 3 para manejar solicitudes HTTP

Ejecute el comando para instalar Axios:

```
npm install axios
```

Axios es un cliente HTTP basado en promesas para el navegador y node.js.

Construye formularios en Vue con Bootstrap 4

Necesitamos almacenar los datos en la aplicación MEVN, por lo que necesitamos crear un formulario vue utilizando el componente Bootstrap Form.

Para ello, abra el archivo “components/CreateComponent.vue” , luego coloque el siguiente código en él.

```
<template>
  <div class="row justify-content-center">
    <div class="col-md-6">
      <h3 class="text-center">Create Student</h3>
      <form @submit.prevent="handleSubmitForm">
        <div class="form-group">
          <label>Name</label>
          <input
            type="text"
            class="form-control"
            v-model="student.name"
            required
          />
        </div>

        <div class="form-group">
          <label>Email</label>
          <input
            type="email"
            class="form-control"
            v-model="student.email"
            required
          />
        </div>

        <div class="form-group">
          <label>Phone</label>
          <input
            type="text"
            class="form-control"
            v-model="student.phone"
            required
          />
```

```
        </div>

        <div class="form-group">
          <button class="btn btn-danger btn-block">Create</button>
        </div>
      </form>
    </div>
  </div>
</template>

<script>
export default {
  data() {
    return {
      student: {
        name: "",
        email: "",
        phone: "",
      },
    };
  },
  methods: {
    handleSubmitForm() {},
  },
};
</script>
```

Creamos un formulario básico con campo de nombre, correo electrónico y número de teléfono. Creamos un formulario usando el componente de formulario Bootstrap.

Configuración del entorno del servidor en node.js

Ahora, necesitamos crear API REST usando Node + Express y MongoDB en la aplicación Vue. Cree una carpeta de backend en la raíz del proyecto Vue.

```
mkdir backend
```

Genere el archivo “package.json” separado para el servidor de nodo.

```
npm init -y
```

Ejecute el comando para instalar las siguientes dependencias para Node/Express js.

```
npm i express body-parser cors mongoose
```

Además, instale un servidor nodemon como dependencia de desarrollo. Para que no necesitemos reiniciar cada vez, cambiamos el código de nuestro servidor.

```
npm install nodemon --save-dev
```

Configurar e iniciar la base de datos MongoDB

Cree el archivo “backend/database.js” en la raíz de su aplicación de nodo. A continuación, agregue el código dado en él.

```
module.exports = {
  db: "mongodb://localhost:27017/vuecrudmevn",
};
```

En este ejercicio, estamos trabajando con MongoDB para almacenar los datos de los estudiantes, por lo que debe configurar MongoDB en su sistema de desarrollo.

Abra la ventana de terminal y ejecute el siguiente comando para iniciar MongoDB en su máquina local.

```
mongod
```

Crear modelo de mongoose

Cree el archivo “/backend/models/Student.js” y pegue el siguiente código dentro del archivo. Definiremos los valores de nombre, correo electrónico y teléfono dentro del modelo Student.

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;

let studentSchema = new Schema(
  {
    name: {
      type: String,
    },
    email: {
      type: String,
    },
    phone: {
      type: Number,
    },
  },
  {
    collection: "students",
  }
);
```

```
module.exports = mongoose.model("Student", studentSchema);
```

Crear ruta en la aplicación Node/Express

Cree un directorio “backend/routes”; aquí, tenemos que crear el archivo “student.route.js” y colocar todo el código que se proporciona a continuación dentro de él.

```
const express = require("express");
const studentRoute = express.Router();

// Student model
let StudentModel = require("../models/Student");

studentRoute.route("/").get((req, res) => {
  StudentModel.find((error, data, next) => {
    if (error) {
      return next(error);
    } else {
      console.log(error);
      res.json(data);
    }
  });
});

studentRoute.route("/create-student").post((req, res, next) => {
  StudentModel.create(req.body, (error, data) => {
    if (error) {
      return next(error);
    } else {
      console.log(data);
      res.json(data);
    }
  });
});

studentRoute.route("/edit-student/:id").get((req, res) => {
  StudentModel.findById(req.params.id, (error, data, next) => {
    if (error) {
      console.log(error);
      return next(error);
    } else {
      res.json(data);
    }
  });
});

// Update student
studentRoute.route("/update-student/:id").put((req, res, next) => {
  StudentModel.findByIdAndUpdate(
```

```
    req.params.id,
    {
      $set: req.body,
    },
    (error, data) => {
      if (error) {
        console.log(error);
        return next(error);
      } else {
        res.json(data);
        console.log("Student successfully updated!");
      }
    }
  );
});

// Delete student
studentRoute.route("/delete-student/:id").delete((req, res, next) => {
  StudentModel.findByIdAndRemove(req.params.id, (error, data) => {
    if (error) {
      return next(error);
    } else {
      res.status(200).json({
        msg: data,
      });
    }
  });
});

module.exports = studentRoute;
```

Definimos las rutas que se comunicarán con el servidor usando la biblioteca Axios. Podemos realizar la operación CRUD usando los métodos GET, POST, UPDATE & DELETE.

Para ello, cree el archivo “backend/app.js” y coloque el siguiente código que contiene la configuración del no de servidor.

```
let express = require("express"),
  cors = require("cors"),
  mongoose = require("mongoose"),
  database = require("./database"),
  bodyParser = require("body-parser");

// Connect mongoDB
mongoose.Promise = global.Promise;
mongoose
  .connect(database.db, {
    useNewUrlParser: true,
    useUnifiedTopology: true,
  })
  .then(
    () => {
```

```
      console.log("Database connected");
    },
    (error) => {
      console.log("Database could't be connected to: " + error);
    }
  );

const studentAPI = require("../backend/routes/student.route");
const app = express();
app.use(bodyParser.json());
app.use(
  bodyParser.urlencoded({
    extended: false,
  })
);
app.use(cors());

// API
app.use("/api", studentAPI);

// Create port
const port = process.env.PORT || 4000;
const server = app.listen(port, () => {
  console.log("Connected to port " + port);
});

// Find 404
app.use((req, res, next) => {
  next(createError(404));
});

// error handler
app.use(function(err, req, res) {
  console.error(err.message);
  if (!err.statusCode) err.statusCode = 500;
  res.status(err.statusCode).send(err.message);
});
```

Además, no olvide registrar el valor “app.js” dentro de la propiedad "main" en el archivo “package.json.” del nodo servidor:

```
{
  "name": "vue-mevn-stack",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
```

```
    "body-parser": "^1.19.0",
    "cors": "^2.8.5",
    "express": "^4.17.1",
    "mongoose": "^5.13.8"
  },
  "devDependencies": {
    "nodemon": "^2.0.3"
  }
}
```

Iniciar aplicación Node/Express

Hemos construido la siguiente API usando Node y Express en Vue.js.

Abra una nueva CMD dentro de la carpeta “backend” e inicie MongoDB:

```
mongod
```

Puede ver el servidor que se ejecuta en http://localhost:4000/api

Ahora, abra una nueva CMD dentro de la carpeta raíz del proyecto y ejecute:

```
npm run serve
```

Puede ver la aplicación vue ejecutándose en http://localhost:8080

Cree datos de estudiantes con Axios POST

El método de publicación de Axios toma la API REST y realiza la solicitud POST al servidor. Crea los datos de los estudiantes que estamos agregando en la base de datos mongoDB.

Una vez que los datos se envían al servidor, puede verificar los datos almacenados en http://localhost:4000/api

Agregue el código que se proporciona a continuación dentro del archivo components/CreateComponent.vue

```
<template>
  <div class="row justify-content-center">
    <div class="col-md-6">
      <h3 class="text-center">Create Student</h3>
      <form @submit.prevent="handleSubmitForm">
        <div class="form-group">
          <label>Name</label>
          <input
            type="text"
            class="form-control"
```

```
            v-model="student.name"
            required
          />
        </div>

        <div class="form-group">
          <label>Email</label>
          <input
            type="email"
            class="form-control"
            v-model="student.email"
            required
          />
        </div>

        <div class="form-group">
          <label>Phone</label>
          <input
            type="text"
            class="form-control"
            v-model="student.phone"
            required
          />
        </div>

        <div class="form-group">
          <button class="btn btn-danger btn-block">Create</button>
        </div>
      </form>
    </div>
  </div>
</template>

<script>
import axios from "axios";

export default {
  data() {
    return {
      student: {
        name: "",
        email: "",
        phone: "",
      },
    };
  },
  methods: {
    handleSubmitForm() {
      let apiURL = "http://localhost:4000/api/create-student";

      axios
        .post(apiURL, this.student)
        .then(() => {
          this.$router.push("/view");
          this.student = {
```

```
            name: "",
            email: "",
            phone: "",
          };
        })
        .catch((error) => {
          console.log(error);
        });
    },
  },
};
</script>
```

Mostrar lista de datos y eliminar datos en Vue

Ahora, mostraremos los datos en forma tabular usando componentes de Bootstrap Table, y haremos la solicitud axios.get () para renderizar los datos del servidor.

Agregue el código que se proporciona a continuación dentro del archivo components/ListComponent.vue

```
<template>
  <div class="row">
    <div class="col-md-12">
      <table class="table table-striped">
        <thead class="thead-dark">
          <tr>
            <th>Name</th>
            <th>Email</th>
            <th>Phone</th>
            <th>Actions</th>
          </tr>
        </thead>
        <tbody>
          <tr v-for="student in Students" :key="student._id">
            <td>{{ student.name }}</td>
            <td>{{ student.email }}</td>
            <td>{{ student.phone }}</td>
            <td>
              <router-link
                :to="{ name: 'edit', params: { id: student._id } }"
                class="btn btn-success"
                >Edit
              </router-link>
              <button
                @click.prevent="deleteStudent(student._id)"
                class="btn btn-danger"
              >
                Delete
              </button>
            </td>
```

```
          </tr>
        </tbody>
      </table>
    </div>
  </div>
</template>

<script>
import axios from "axios";

export default {
  data() {
    return {
      Students: [],
    };
  },
  created() {
    let apiURL = "http://localhost:4000/api";
    axios
      .get(apiURL)
      .then((res) => {
        this.Students = res.data;
      })
      .catch((error) => {
        console.log(error);
      });
  },
  methods: {
    deleteStudent(id) {
      let apiURL = `http://localhost:4000/api/delete-student/${id}`;
      let indexOfArrayItem = this.Students.findIndex((i) => i._id === id);

      if (window.confirm("Do you really want to delete?")) {
        axios
          .delete(apiURL)
          .then(() => {
            this.Students.splice(indexOfArrayItem, 1);
          })
          .catch((error) => {
            console.log(error);
          });
      }
    },
  },
};
</script>

<style>
.btn-success {
  margin-right: 10px;
}
</style>
```

Para eliminar el objeto de estudiante de la base de datos, definimos la función deleteStudent () y la vinculamos al evento de clic con un parámetro de identificación.

La apiURL contiene la API api/delete-student/:id , para averiguar el índice del índice de datos del estudiante en el que se hizo clic de la matriz de Estudiantes, tomamos la ayuda del método findIndex().

Actualizar datos con solicitud POST

Agregue el siguiente código dentro del archivo components/EditComponent.vue

```
<template>
  <div class="row justify-content-center">
    <div class="col-md-6">
      <h3 class="text-center">Update Student</h3>
      <form @submit.prevent="handleUpdateForm">
        <div class="form-group">
          <label>Name</label>
          <input
            type="text"
            class="form-control"
            v-model="student.name"
            required
          />
        </div>

        <div class="form-group">
          <label>Email</label>
          <input
            type="email"
            class="form-control"
            v-model="student.email"
            required
          />
        </div>

        <div class="form-group">
          <label>Phone</label>
          <input
            type="text"
            class="form-control"
            v-model="student.phone"
            required
          />
        </div>

        <div class="form-group">
          <button class="btn btn-danger btn-block">Update</button>
        </div>
      </form>
    </div>
  </div>
```

```
</template>

<script>
import axios from "axios";

export default {
  data() {
    return {
      student: {},
    };
  },
  created() {
    let apiURL = `http://localhost:4000/api/edit-
student/${this.$route.params.id}`;

    axios.get(apiURL).then((res) => {
      this.student = res.data;
    });
  },
  methods: {
    handleUpdateForm() {
      let apiURL = `http://localhost:4000/api/update-
student/${this.$route.params.id}`;

      axios
        .put(apiURL, this.student)
        .then((res) => {
          console.log(res);
          this.$router.push("/view");
        })
        .catch((error) => {
          console.log(error);
        });
    },
  },
};
</script>
```