1. Authentication using laravel breeze api and spatie for roles and permission.
2. Create a crud operation using restful api.
3. **Create a contact us form and send email to admin with name,subject,phone number and message.**
4. **List all the users with his status i.e. online or offline.**
5. **Login with laravel Socialite.**
6. **Write a logic to export database data to pdf,excel and csv file. Also write logic to import excel file to insert into database table.**

# **Authentication using laravel breeze api and vuejs.**

# **Laravel Backend:**

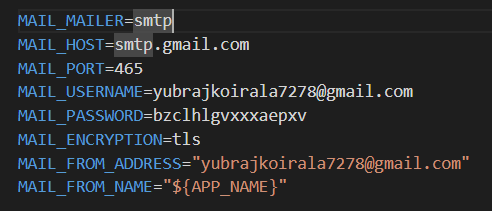




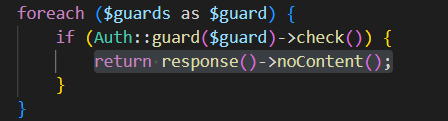
Inside .env







Inside RedirectIfAuthenticated.php



# **Vuejs Frontend:**

1. Install necessary packages:











Clear App.Vue and delete HelloWorld.vue and delete files inside assets.

Create src/router/index.js

1. Change vuejs port to 3000 from vite.config.js



1. Create src/axios.js

import axios from "axios";

// Base configuration

axios.defaults.baseURL = "http://localhost:8000";

axios.defaults.withCredentials = true;

// Interceptor to add CSRF token header

axios.interceptors.request.use(config => {

const token = document.cookie.split('; ').find(row => row.startsWith('XSRF-TOKEN='));

if (token) {

config.headers['X-XSRF-TOKEN'] = decodeURIComponent(token.split('=')[1]);

}

return config;

});

export default axios;

1. Inside main.js

import { createApp } from 'vue'

import './style.css'

import App from './App.vue'

import './axios'

import { createPinia } from 'pinia'

import router from './router'

const pinia = createPinia();

const app=createApp(App);

app.use(pinia)

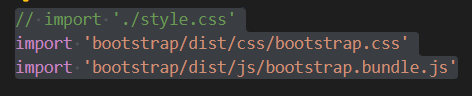
app.use(router)

app.mount('#app')

1. How to use bootstrap in vuejs



Inside main.js



1. How to use bootstrap icon in vuejs



Inside main.js



1. How to use sweetalert in vuejs



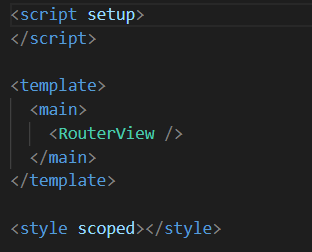
1. How to use toastify in vuejs



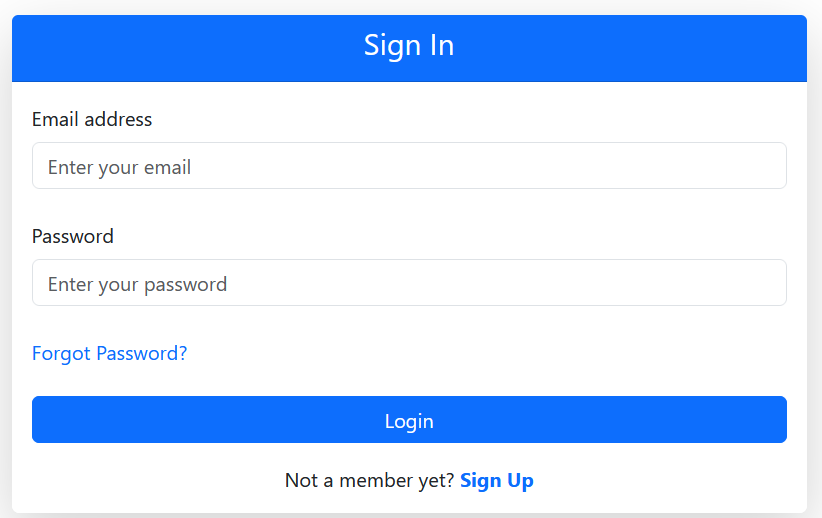
Inside main.js

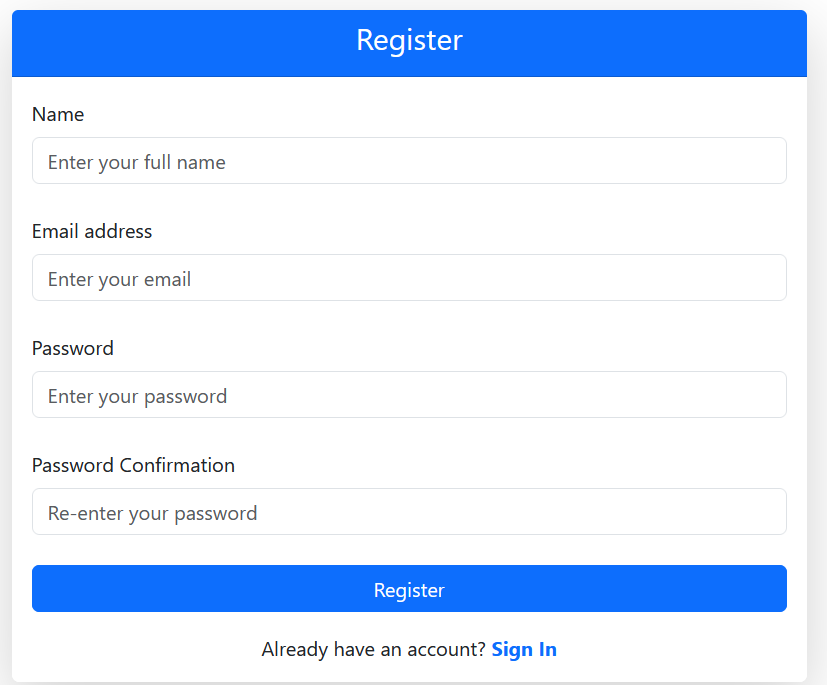


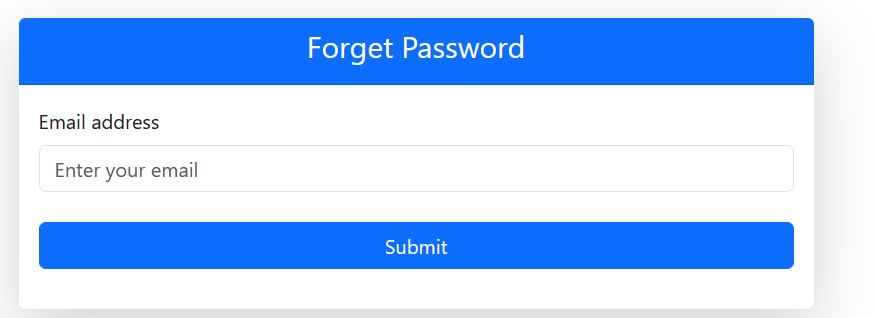
1. Inside App.vue

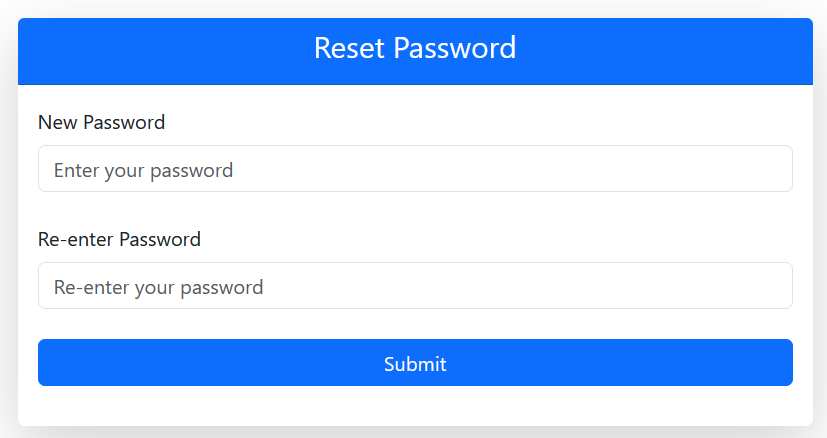


1. Create src/components/frontend/Home.vue,Nav.vue, || src/components/auth Login.vue, Register.vue,ForgetPassword.vue ,ResetPassword.vue and src/components/backend/Dashboard.vue and also src/components/layouts/AuthLayout.vue, src/components/layouts/FrontendLayout.vue and src/components/layouts/BackendLayout.vue





****

****

1. **Inside stores/auth.js**

import { defineStore } from "pinia";

import { ref } from "vue";

import axios from "axios";

import router from "../router";

export const useAuthStore = defineStore("auth", () => {

  // ========= State =========

  const authUser = ref(null); // Holds the authenticated user data

  const authErrors = ref([]); // Holds validation or other errors

  const authStatus = ref(null);

  const loading = ref(false);

  //  =======Fetch the CSRF token==============

  const getToken = async () => {

    try {

      await axios.get("/sanctum/csrf-cookie");

    } catch (error) {

      console.error("Error fetching CSRF token:", error);

    }

  };

  // ====Handle user login===============

  const handleLogin = async (credentials) => {

    try {

      loading.value = true;

      await getToken(); // Ensure CSRF token is set

      const response = await axios.post("/login", credentials);

      authUser.value = response.data.user;

      router.push("/admin/home");

    } catch (error) {

      if (error.response && error.response.status === 422) {

        authErrors.value = error.response.data.errors;

      }

    } finally {

      loading.value = false;

    }

  };

  //  ==========Handle user registration================

  const handleRegister = async (userDetails) => {

    try {

      loading.value = true;

      await getToken(); // Ensure CSRF token is set

      const response = await axios.post("/register", userDetails);

      authUser.value = response.data.user;

      router.push("/admin/home");

    } catch (error) {

      if (error.response && error.response.status === 422) {

        authErrors.value = error.response.data.errors;

      }

    } finally {

      loading.value = false;

    }

  };

  // =======Fetch authenticated user's information==========

  const getUser = async () => {

    try {

      await getToken(); // Ensure CSRF token is set

      const { data } = await axios.get("/api/user");

      authUser.value = data;

    } catch (error) {

      authUser.value = null;

    }

  };

  // =======Handle user logout=============

  const handleLogout = async () => {

    try {

      await axios.post("/logout");

      authUser.value = null;

      router.push("/login");

    } catch (error) {

      console.error("Logout failed:", error);

    }

  };

  // =====Handle forgot password============

  const handleForgetPassword = async (email) => {

    try {

      loading.value = true;

      await getToken(); // Ensure CSRF token is set

      const response = await axios.post("/forgot-password", { email });

      authStatus.value = response.data.status;

    } catch (error) {

      if (error.response && error.response.status === 422) {

        authErrors.value = error.response.data.errors;

      }

    } finally {

      loading.value = false;

    }

  };

  // =========Handle reset password=============

  const handleResetPassword = async (resetData) => {

    try {

      loading.value = true;

      await getToken(); // Ensure CSRF token is set

      const response = await axios.post("/reset-password", resetData);

      // authUser.value = response.data.user;

      authStatus.value = response.data.status;

      router.push("/login");

    } catch (error) {

      if (error.response && error.response.status === 422) {

        console.log(error.response.data)

        authErrors.value = error.response.data.errors;

      }

    } finally {

      loading.value = false;

    }

  };

  // ===========reset properties===========

  const resetFields = () => {

    authErrors.value = [];

    authStatus.value = null;

    loading.value = false;

  };

  return {

    authUser,

    authErrors,

    getUser,

    handleLogin,

    handleRegister,

    handleLogout,

    handleForgetPassword,

    resetFields,

    handleResetPassword,

    authStatus,

    loading,

  };

});

1. **Inside router/index.js**

import { createRouter, createWebHistory } from "vue-router";

import { useAuthStore } from "../stores/auth";

const routes = [

  {

    path: "/",

    name: "Frontend",

    component: () => import("../components/layouts/FrontendLayout.vue"),

    children: [

      {

        path: "",

        name: "Home",

        component: () => import("../components/frontend/Home.vue"),

      },

    ],

  },

  {

    path: "/auth",

    name: "Auth",

    component: () => import("../components/layouts/AuthLayout.vue"),

    children: [

      {

        path: "/login",

        name: "Login",

        component: () => import("../components/auth/Login.vue"),

        meta: { guestOnly: true },

      },

      {

        path: "/register",

        name: "Register",

        component: () => import("../components/auth/Register.vue"),

        meta: { guestOnly: true },

      },

      {

        path: "/forget-password",

        name: "ForgetPassword",

        component: () => import("../components/auth/ForgetPassword.vue"),

        meta: { guestOnly: true },

      },

      {

        path: "/password-reset/:token",

        name: "ResetPassword",

        component: () => import("../components/auth/ResetPassword.vue"),

        meta: { guestOnly: true },

      },

    ],

  },

  {

    path: "/admin",

    name: "Admin",

    component: () => import("../components/layouts/BackendLayout.vue"),

    children: [

      {

        path: "home",

        name: "Dashboard",

        component: () => import("../components/backend/Dashboard.vue"),

        meta: { requiresAuth: true },

      },

    ],

  },

];

const router = createRouter({

  history: createWebHistory(),

  routes,

});

// Global Navigation Guard

router.beforeEach(async (to, from, next) => {

  const authStore = useAuthStore(); // Access the auth store

  // Fetch the authenticated user if not already loaded

  if (!authStore.authUser) {

    await authStore.getUser(); // Fetch the user information from the backend

  }

  // Handle protected routes

  if (to.meta.requiresAuth && !authStore.authUser) {

    // Redirect to login if the user is not authenticated

    return next({ name: "Login" });

  }

  // Handle guest-only routes (e.g., /login, /register)

  if (to.meta.guestOnly && authStore.authUser) {

    // Redirect authenticated users to the dashboard

    return next({ name: "Dashboard" });

  }

  // Reset the fields on every route change

  authStore.resetFields();

  // Allow navigation

  next();

});

export default router;

1. **Inside FrontendLayout.vue, BackendLayout.vue and AuthLayout.vue**

<script setup>

import Nav from "../frontend/Nav.vue";

</script>

<template>

    <main>

      <Nav/>

      <!-- The child routes will be injected here -->

      <router-view />

    </main>

</template>

1. **Inside Login.vue**

<script setup>

import { ref } from "vue";

import { useAuthStore } from "../../stores/auth";

// Access the authentication store

const authStore = useAuthStore();

// properties

const form = ref({

  email: "",

  password: "",

});

</script>

<template>

  <div class="container py-5">

    <div class="row justify-content-center">

      <div class="col-md-6">

        <div class="card shadow-lg border-0">

          <div class="card-header bg-primary text-white text-center">

            <h4>Sign In</h4>

          </div>

          <div class="card-body">

            <form @submit.prevent="authStore.handleLogin(form)">

              <div class="mb-4">

                <label for="email" class="form-label">Email address</label>

                <input

                  type="email"

                  v-model="form.email"

                  class="form-control"

                  id="email"

                  placeholder="Enter your email"

                />

                <span class="text-danger" v-if="authStore.authErrors.email">{{

                  authStore.authErrors.email[0]

                }}</span>

              </div>

              <div class="mb-4">

                <label for="exampleInputPassword1" class="form-label"

                  >Password</label

                >

                <input

                  type="password"

                  v-model="form.password"

                  class="form-control"

                  id="exampleInputPassword1"

                  placeholder="Enter your password"

                />

                <span

                  class="text-danger"

                  v-if="authStore.authErrors.password"

                  >{{ authStore.authErrors.password[0] }}</span

                >

              </div>

              <div class="d-flex justify-content-between mb-4">

                <router-link

                  :to="{ name: 'ForgetPassword' }"

                  class="text-decoration-none"

                  >Forgot Password?</router-link

                >

              </div>

              <button

                type="submit"

                class="btn btn-primary w-100 mb-3"

                :disabled="authStore.loading"

              >

                {{ authStore.loading ? "Loading..." : "Login" }}

              </button>

            </form>

            <div class="text-center">

              <p class="mb-0">

                Not a member yet?

                <router-link

                  :to="{ name: 'Register' }"

                  class="text-decoration-none text-primary fw-bold"

                  >Sign Up</router-link

                >

              </p>

            </div>

          </div>

        </div>

      </div>

    </div>

  </div>

</template>

1. **Inside Register.vue**

<script setup>

import { ref } from "vue";

import { useAuthStore } from "../../stores/auth";

const authStore = useAuthStore();

const form = ref({

  name: "",

  email: "",

  password: "",

  password\_confirmation: "",

});

</script>

<template>

  <div class="container py-5">

    <div class="row justify-content-center">

      <div class="col-md-6">

        <div class="card shadow-lg border-0">

          <div class="card-header bg-primary text-white text-center">

            <h4>Register</h4>

          </div>

          <div class="card-body">

            <form @submit.prevent="authStore.handleRegister(form)">

              <div class="mb-4">

                <label for="name" class="form-label">Name</label>

                <input

                  type="text"

                  class="form-control"

                  id="name"

                  placeholder="Enter your full name"

                  v-model="form.name"

                />

                <span class="text-danger" v-if="authStore.authErrors.name">{{

                  authStore.authErrors.name[0]

                }}</span>

              </div>

              <div class="mb-4">

                <label for="email" class="form-label">Email address</label>

                <input

                  type="email"

                  class="form-control"

                  id="email"

                  placeholder="Enter your email"

                  v-model="form.email"

                />

                <span class="text-danger" v-if="authStore.authErrors.email">{{

                  authStore.authErrors.email[0]

                }}</span>

              </div>

              <div class="mb-4">

                <label for="password" class="form-label">Password</label>

                <input

                  type="password"

                  class="form-control"

                  id="password"

                  placeholder="Enter your password"

                  v-model="form.password"

                />

                <span

                  class="text-danger"

                  v-if="authStore.authErrors.password"

                  >{{ authStore.authErrors.password[0] }}</span

                >

              </div>

              <div class="mb-4">

                <label for="password\_confirmation" class="form-label"

                  >Password Confirmation</label

                >

                <input

                  type="password"

                  class="form-control"

                  id="password\_confirmation"

                  placeholder="Re-enter your password"

                  v-model="form.password\_confirmation"

                />

              </div>

              <button

                type="submit"

                class="btn btn-primary w-100 mb-3"

                :disabled="authStore.loading"

              >

                {{ authStore.loading ? "Loading..." : "Register" }}

              </button>

            </form>

            <div class="text-center">

              <p class="mb-0">

                Already have an account?

                <router-link

                  :to="{ name: 'Login' }"

                  class="text-decoration-none text-primary fw-bold"

                  >Sign In</router-link

                >

              </p>

            </div>

          </div>

        </div>

      </div>

    </div>

  </div>

</template>

<style scoped></style>

1. **ForgetPassword.vue**

<script setup>

import { ref } from "vue";

import { useAuthStore } from "../../stores/auth"; // Adjust the path based on your project structure

// Access the auth store

const authStore = useAuthStore();

// Reactive properties

const email = ref("");

// Handle the forget password form submission

const handleForgetPassword = async () => {

  // Call the store method to handle the forgot password logic

  await authStore.handleForgetPassword(email.value);

  // Reset email and display success message

  email.value = "";

};

</script>

<template>

  <div class="container py-5">

    <div class="row justify-content-center">

      <div class="col-md-6">

        <div

          class="alert alert-success"

          role="alert"

          v-if="authStore.authStatus"

        >

          {{ authStore.authStatus }}

        </div>

        <div class="card shadow-lg border-0">

          <div class="card-header bg-primary text-white text-center">

            <h4>Forget Password</h4>

          </div>

          <div class="card-body">

            <form @submit.prevent="handleForgetPassword">

              <div class="mb-4">

                <label for="email" class="form-label">Email address</label>

                <input

                  type="email"

                  v-model="email"

                  class="form-control"

                  id="email"

                  placeholder="Enter your email"

                />

                <!-- Display validation error for email -->

                <span class="text-danger" v-if="authStore.authErrors.email">{{

                  authStore.authErrors.email[0]

                }}</span>

              </div>

              <button

                type="submit"

                class="btn btn-primary w-100 mb-3"

                :disabled="authStore.loading"

              >

                {{ authStore.loading ? "Loading..." : "Submit" }}

              </button>

            </form>

          </div>

        </div>

      </div>

    </div>

  </div>

</template>

1. **Inside ResetPassword.vue**

<script setup>

import { ref } from "vue";

import { useAuthStore } from "../../stores/auth";

import { useRoute } from "vue-router";

const authStore = useAuthStore();

const route = useRoute();

// Reactive properties

const form = ref({

  password: "",

  password\_confirmation: "",

  email: route.query.email,

  token: route.params.token,

});

</script>

<template>

  <div class="container py-5">

    <div class="row justify-content-center">

      <div class="col-md-6">

        <div

          class="alert alert-success"

          role="alert"

          v-if="authStore.authStatus"

        >

          {{ authStore.authStatus }}

        </div>

        <div class="card shadow-lg border-0">

          <div class="card-header bg-primary text-white text-center">

            <h4>Reset Password</h4>

          </div>

          <div class="card-body">

            <form @submit.prevent="authStore.handleResetPassword(form)">

              <div class="mb-4">

                <label for="password" class="form-label">New Password</label>

                <input

                  type="password"

                  v-model="form.password"

                  class="form-control"

                  id="password"

                  placeholder="Enter your password"

                />

                <!-- Display validation error for password -->

                <span

                  class="text-danger"

                  v-if="authStore.authErrors.password"

                  >{{ authStore.authErrors.password[0] }}</span

                >

                <span class="text-danger" v-if="authStore.authErrors.email">{{

                  authStore.authErrors.email[0]

                }}</span>

              </div>

              <div class="mb-4">

                <label for="password\_confirmation" class="form-label"

                  >Re-enter Password</label

                >

                <input

                  type="password"

                  v-model="form.password\_confirmation"

                  class="form-control"

                  id="password\_confirmation"

                  placeholder="Re-enter your password"

                />

              </div>

              <button

                type="submit"

                class="btn btn-primary w-100 mb-3"

                :disabled="authStore.loading"

              >

                {{ authStore.loading ? "Loading..." : "Submit" }}

              </button>

            </form>

          </div>

        </div>

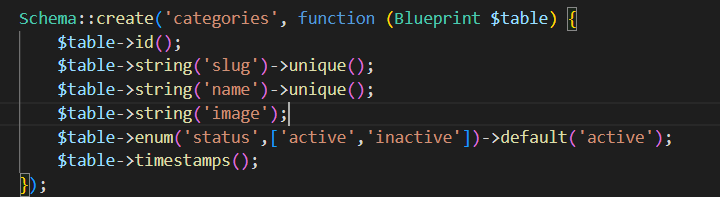
      </div>

    </div>

  </div>

</template>

1. **Create a crud operation using restful api.**
2. Make model and migration



1. Make controller php artisan make:controller Api/CategoryController --model=Category

 /\*\*

     \* Display a listing of the resource.

     \*/

    public function index(Request $request)

    {

        $perPage = $request->query('per\_page', 10);

        $search = $request->query('search', '');

        $sortBy = $request->query('sort\_by', 'created\_at'); // Default to created\_at

        $sortDirection = $request->query('sort\_direction', 'desc'); // Default to desc

        $query = Category::query();

        if ($search) {

            $query->where(function ($q) use ($search) {

                $q->where('name', 'like', '%' . $search . '%')

                    ->orWhere('status', 'like', '%' . $search . '%');

            });

        }

        // Validate sort\_by to prevent SQL injection; only allow specific fields

        $allowedSorts = ['name', 'status', 'created\_at'];

        $sortBy = in\_array($sortBy, $allowedSorts) ? $sortBy : 'created\_at';

        $sortDirection = in\_array(strtolower($sortDirection), ['asc', 'desc']) ? $sortDirection : 'desc';

        $query->orderBy($sortBy, $sortDirection);

        $categories = $query->paginate($perPage);

        return CategoryResource::collection($categories);

    }

    /\*\*

     \* Store a newly created resource in storage.

     \*/

    public function store(Request $request)

    {

        $validator = Validator::make($request->all(), [

            'name' => 'required|string|max:255',

            'status' => 'required|in:active,inactive',

            'image' => 'required|image|max:2048', // Max 2MB

        ]);

        if ($validator->fails()) {

            return response()->json([

                'errors' => $validator->errors()

            ], 422);

        }

        $data = $request->all();

        if ($request->hasFile('image')) {

            $data['image'] = $request->file('image')->store('categories', 'public');

        }

        $category = Category::create($data);

        return new CategoryResource($category);

    }

    /\*\*

     \* Display the specified resource.

     \*/

    public function show(Category $category)

    {

        return new CategoryResource($category);

    }

    /\*\*

     \* Update the specified resource in storage.

     \*/

    public function update(Request $request, Category $category)

    {

        $validator = Validator::make($request->all(), [

            'name' => 'sometimes|required|string|max:255',

            'status' => 'sometimes|required|in:active,inactive',

            'image' => 'nullable|image|max:2048',

        ]);

        if ($validator->fails()) {

            return response()->json([

                'errors' => $validator->errors()

            ], 422);

        }

        $data = $request->all();

        if ($request->hasFile('image')) {

            if ($category->image) {

                Storage::disk('public')->delete($category->image);

            }

            $data['image'] = $request->file('image')->store('categories', 'public');

        }

        $category->update($data);

        return new CategoryResource($category);

    }

    /\*\*

     \* Remove the specified resource from storage.

     \*/

    public function destroy(Category $category)

    {

        if ($category->image) {

            Storage::disk('public')->delete($category->image);

        }

        $category->delete();

        return response()->json(null, 204);

    }

1. Make resource php artisan make:resource CategoryResource

 public function toArray(Request $request): array

    {

        return [

            'id' => $this->id,

            'slug' => $this->slug,

            'name' => $this->name,

            'status' => $this->status,

            'image' => $this->image ? asset('storage/' . $this->image) : null,

            'created\_at' => $this->created\_at->toDateTimeString(),

            'updated\_at' => $this->updated\_at->toDateTimeString(),

        ];

    }

1. Inside api.php

Route::group(['prefix' => 'categories'], function () {

    Route::get('/', [CategoryController::class, 'index'])->name('categories.index');

    Route::post('/', [CategoryController::class, 'store'])->name('categories.store');

    Route::get('/{category}', [CategoryController::class, 'show'])->name('categories.show');

    Route::put('/{category}', [CategoryController::class, 'update'])->name('categories.update');

    Route::delete('/{category}', [CategoryController::class, 'destroy'])->name('categories.destroy');

});

**In VueJS**

1. Inside src/axios.js

import axios from "axios";

// Base configuration

axios.defaults.baseURL = "http://localhost:8000";

axios.defaults.withCredentials = true;

// Interceptor to add CSRF token header

axios.interceptors.request.use(config => {

    const token = document.cookie.split('; ').find(row => row.startsWith('XSRF-TOKEN='));

    if (token) {

        config.headers['X-XSRF-TOKEN'] = decodeURIComponent(token.split('=')[1]);

    }

    return config;

});

export default axios;

1. Inside src/stores/category.js

// src/stores/category.js

import { defineStore } from "pinia";

import { ref } from "vue";

import axios from "axios"; // Assuming your axios configuration is here

export const useCategoryStore = defineStore("category", () => {

    // ========= State =========

    const categories = ref([]); // List of all categories

    const currentCategory = ref(null); // Single category for view/edit

    const categoryErrors = ref([]); // Validation or API errors

    const loading = ref(false); // Loading state for async operations

    const pagination = ref({}); // Pagination metadata

    // ======= Fetch all categories (View/List) =======

    const fetchCategories = async (page = 1, perPage = 10, search = '', sortBy = 'created\_at', sortDirection = 'desc') => {

        console.log(sortDirection);

        try {

            loading.value = true;

            const url = `/api/categories?page=${page}&per\_page=${perPage}` +

                (search ? `&search=${encodeURIComponent(search)}` : '') +

                `&sort\_by=${sortBy}&sort\_direction=${sortDirection}`;

            const response = await axios.get(url);

            console.log('API Response:', response.data);

            categories.value = response.data.data;

            pagination.value = response.data.meta || {};

            console.log('Pagination:', pagination.value);

        } catch (error) {

            console.error("Fetch error:", error);

            categories.value = [];

            pagination.value = {};

        } finally {

            loading.value = false;

        }

    };

    // ======= Fetch a single category (View/Edit) =======

    const fetchCategory = async (slug) => {

        try {

            loading.value = true;

            const response = await axios.get(`/api/categories/${slug}`);

            currentCategory.value = response.data.data;

        } catch (error) {

            console.error("Error fetching category:", error);

            categoryErrors.value = error.response?.data?.errors || ["Category not found"];

            currentCategory.value = null;

        } finally {

            loading.value = false;

        }

    };

    // ======= Store/Create a new category =======

    const storeCategory = async (categoryData) => {

        try {

            loading.value = true;

            categoryErrors.value = [];

            const response = await axios.post("/api/categories", categoryData, {

                headers: { "Content-Type": "multipart/form-data" },

            });

            const newCategory = response.data.data;

            return newCategory; // Return new category for local update

        } catch (error) {

            if (error.response && error.response.status === 422) {

                categoryErrors.value = error.response.data.errors;

            } else {

                console.error("Error creating category:", error);

            }

            throw error;

        } finally {

            loading.value = false;

        }

    };

    // ======= Update an existing category =======

    const updateCategory = async (slug, categoryData) => {

        try {

            loading.value = true;

            categoryErrors.value = [];

            categoryData.append('\_method', 'PUT');

            const response = await axios.post(`/api/categories/${slug}`, categoryData, {

                headers: { "Content-Type": "multipart/form-data" },

            });

            const updatedCategory = response.data.data;

            return updatedCategory; // Return updated category for local update

        } catch (error) {

            if (error.response && error.response.status === 422) {

                categoryErrors.value = error.response.data.errors;

            } else {

                console.error("Error updating category:", error);

            }

            throw error;

        } finally {

            loading.value = false;

        }

    };

    // ======= Delete a category =======

    const deleteCategory = async (slug) => {

        try {

            loading.value = true;

            await axios.delete(`/api/categories/${slug}`);

            return slug; // Return slug for local removal

        } catch (error) {

            console.error("Error deleting category:", error);

            throw error;

        } finally {

            loading.value = false;

        }

    };

    // ======= Reset errors =======

    const resetErrors = () => {

        categoryErrors.value = [];

    };

    // ======= Return state and actions =======

    return {

        categories,

        currentCategory,

        categoryErrors,

        loading,

        pagination,

        fetchCategories,

        fetchCategory,

        storeCategory,

        updateCategory,

        deleteCategory,

        resetErrors,

    };

});

1. Inside src/components/backend/category/Category.vue

<script setup>

import { ref, onMounted, computed, watch } from 'vue';

import { useCategoryStore } from '../../../stores/category';

import CreateCategoryModal from './CreateCategoryModal.vue';

import UpdateCategoryModal from './UpdateCategoryModal.vue';

import Swal from 'sweetalert2';

import \* as bootstrap from 'bootstrap';

const debounce = (fn, delay) => {

    let timeoutId;

    return (...args) => {

        clearTimeout(timeoutId);

        timeoutId = setTimeout(() => fn(...args), delay);

    };

};

const categoryStore = useCategoryStore();

const createModalRef = ref(null);

const updateModalRef = ref(null);

const selectedSlug = ref(null);

const searchQuery = ref('');

const selectedImage = ref(null);

const sortBy = ref('created\_at');

const sortDirection = ref('desc');

const currentPage = ref(1);

const itemsPerPage = ref(10);

const perPageOptions = [5, 10, 20, 50];

const tableItems = computed(() => {

    const startIndex = (currentPage.value - 1) \* itemsPerPage.value + 1;

    return categoryStore.categories.map((category, index) => ({

        ...category,

        sn: startIndex + index,

    }));

});

const totalItems = computed(() => categoryStore.pagination.total || 0);

const totalPages = computed(() => Math.ceil(totalItems.value / itemsPerPage.value));

onMounted(() => {

    fetchCategories();

});

const fetchCategories = () => {

    categoryStore.fetchCategories(

        currentPage.value,

        itemsPerPage.value,

        searchQuery.value,

        sortBy.value,

        sortDirection.value

    );

};

const debouncedFetchCategories = debounce(fetchCategories, 1000);

watch(searchQuery, () => {

    currentPage.value = 1;

    debouncedFetchCategories();

});

const toggleSort = (field) => {

    if (sortBy.value === field) {

        sortDirection.value = sortDirection.value === 'asc' ? 'desc' : 'asc';

    } else {

        sortBy.value = field;

        sortDirection.value = 'asc';

    }

    currentPage.value = 1;

    fetchCategories();

};

const goToPreviousPage = () => {

    if (currentPage.value > 1) {

        currentPage.value--;

        fetchCategories();

    }

};

const goToNextPage = () => {

    if (currentPage.value < totalPages.value) {

        currentPage.value++;

        fetchCategories();

    }

};

const changePerPage = (event) => {

    itemsPerPage.value = parseInt(event.target.value, 10);

    currentPage.value = 1;

    fetchCategories();

};

const openCreateModal = () => {

    createModalRef.value.open();

};

const openUpdateModal = (slug) => {

    selectedSlug.value = slug;

    if (updateModalRef.value) {

        updateModalRef.value.open(slug);

    } else {

        console.error('Update modal reference is null');

    }

};

const handleCreateCategory = (newCategory) => {

    if (categoryStore.categories.length < itemsPerPage.value) {

        categoryStore.categories.unshift(newCategory);

        categoryStore.pagination.total = (categoryStore.pagination.total || 0) + 1;

        if (categoryStore.categories.length > itemsPerPage.value) {

            categoryStore.categories.pop();

        }

    } else {

        fetchCategories();

    }

};

const handleUpdateCategory = (updatedCategory) => {

    const index = categoryStore.categories.findIndex(cat => cat.slug === updatedCategory.slug);

    if (index !== -1) {

        categoryStore.categories[index] = updatedCategory;

    }

};

const confirmDelete = async (slug) => {

    const result = await Swal.fire({

        title: 'Are you sure?',

        text: "You won't be able to revert this!",

        icon: 'warning',

        showCancelButton: true,

        confirmButtonColor: '#3085d6',

        cancelButtonColor: '#d33',

        confirmButtonText: 'Yes, delete it!',

        cancelButtonText: 'No, cancel',

    });

    if (result.isConfirmed) {

        try {

            const deletedSlug = await categoryStore.deleteCategory(slug);

            categoryStore.categories = categoryStore.categories.filter(cat => cat.slug !== deletedSlug);

            categoryStore.pagination.total = (categoryStore.pagination.total || 1) - 1;

            if (!categoryStore.categories.length && currentPage.value > 1) {

                currentPage.value--;

                fetchCategories();

            } else if (categoryStore.categories.length === 0 && currentPage.value === 1) {

                fetchCategories();

            }

            Swal.fire({

                title: 'Deleted!',

                text: 'The category has been deleted.',

                icon: 'success',

                timer: 1500,

                showConfirmButton: false,

            });

        } catch (error) {

            Swal.fire({

                title: 'Error!',

                text: 'Failed to delete the category.',

                icon: 'error',

                confirmButtonText: 'OK',

            });

        }

    }

};

const openImageModal = (imageUrl) => {

    selectedImage.value = imageUrl;

    const modalElement = document.getElementById('imageModal');

    if (modalElement) {

        const modal = new bootstrap.Modal(modalElement);

        modal.show();

    } else {

        console.error('Modal element not found');

    }

};

</script>

<template>

    <main class="container-fluid py-4 px-5 bg-light">

        <div class="card border-0 shadow-sm">

            <div class="card-header bg-white py-3 px-4 d-flex justify-content-between align-items-center">

                <h5 class="mb-0 fw-semibold">Recent Categories</h5>

                <div class="d-flex gap-2">

                    <button class="btn btn-primary btn-sm d-flex align-items-center gap-1" @click="openCreateModal">

                        <i class="bi bi-plus"></i> Add New

                    </button>

                    <a href="#" class="btn btn-outline-primary btn-sm d-flex align-items-center gap-1">

                        View all <i class="bi bi-arrow-right"></i>

                    </a>

                </div>

            </div>

            <div class="card-body p-0">

                <!-- Search Bar (Always Visible) -->

                <div class="p-3">

                    <input v-model="searchQuery" type="text" class="form-control" placeholder="Search categories..." />

                </div>

                <!-- Table Content -->

                <div v-if="categoryStore.loading" class="text-center py-4">

                    <div class="spinner-border text-primary" role="status">

                        <span class="visually-hidden">Loading...</span>

                    </div>

                </div>

                <div v-else-if="!categoryStore.categories.length" class="text-center py-4">

                    No categories found.

                </div>

                <div v-else>

                    <div class="table-responsive">

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

                            <thead class="table-light">

                                <tr>

                                    <th scope="col">S.N.</th>

                                    <th scope="col">Image</th>

                                    <th scope="col" class="cursor-pointer" @click="toggleSort('name')">

                                        Name

                                        <!-- <i  class="bi bi-arrow-down-up ms-1"></i>  -->

                                        <i v-if="sortBy === 'name'"

                                            :class="sortDirection === 'asc' ? 'bi bi-arrow-up' : 'bi bi-arrow-down'"

                                            class="ms-1"></i>

                                        <i v-else class="bi bi-arrow-down-up ms-1"></i>

                                    </th>

                                    <th scope="col" class="cursor-pointer" @click="toggleSort('status')">

                                        Status

                                        <!-- <i  class="bi bi-arrow-down-up ms-1"></i> -->

                                        <i v-if="sortBy === 'status'"

                                            :class="sortDirection === 'asc' ? 'bi bi-arrow-up' : 'bi bi-arrow-down'"

                                            class="ms-1"></i>

                                        <i v-else class="bi bi-arrow-down-up ms-1"></i>

                                    </th>

                                    <th scope="col">Actions</th>

                                </tr>

                            </thead>

                            <tbody>

                                <tr v-for="item in tableItems" :key="item.id">

                                    <td>{{ item.sn }}</td>

                                    <td>

                                        <img v-if="item.image" :src="item.image" alt="Category Image"

                                            class="img-thumbnail" style="height: 20px; cursor: pointer;"

                                            @click="openImageModal(item.image)">

                                        <span v-else>-</span>

                                    </td>

                                    <td>{{ item.name }}</td>

                                    <td>

                                        <span :class="[

                                            'badge',

                                            item.status === 'active' ? 'bg-success' : 'bg-danger'

                                        ]">

                                            {{ item.status }}

                                        </span>

                                    </td>

                                    <td>

                                        <button class="btn btn-sm btn-primary me-2" @click="openUpdateModal(item.slug)">

                                            <i class="bi bi-pencil"></i>

                                        </button>

                                        <button class="btn btn-sm btn-danger" @click="confirmDelete(item.slug)">

                                            <i class="bi bi-trash"></i>

                                        </button>

                                    </td>

                                </tr>

                            </tbody>

                        </table>

                    </div>

                    <div class="d-flex justify-content-between align-items-center p-3">

                        <div>

                            <select v-model="itemsPerPage" class="form-select form-select-sm" @change="changePerPage">

                                <option v-for="option in perPageOptions" :key="option" :value="option">

                                    {{ option }}

                                </option>

                            </select>

                        </div>

                        <div class="text-muted">

                            Showing {{ (currentPage - 1) \* itemsPerPage + 1 }} to

                            {{ Math.min(currentPage \* itemsPerPage, totalItems) }} of {{ totalItems }} entries

                        </div>

                        <div>

                            <button class="btn btn-sm btn-primary me-2" :disabled="currentPage <= 1"

                                @click="goToPreviousPage">

                                Previous

                            </button>

                            <span>Page {{ currentPage }} of {{ totalPages }}</span>

                            <button class="btn btn-sm btn-primary ms-2" :disabled="currentPage >= totalPages"

                                @click="goToNextPage">

                                Next

                            </button>

                        </div>

                    </div>

                </div>

            </div>

        </div>

        <div class="modal fade" id="imageModal" tabindex="-1" aria-labelledby="imageModalLabel" aria-hidden="true">

            <div class="modal-dialog modal-dialog-centered">

                <div class="modal-content">

                    <div class="modal-header">

                        <h5 class="modal-title" id="imageModalLabel">Category Image</h5>

                        <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>

                    </div>

                    <div class="modal-body text-center">

                        <img v-if="selectedImage" :src="selectedImage" alt="Full Image" class="img-fluid"

                            style="max-height: 500px;">

                    </div>

                    <div class="modal-footer">

                        <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Close</button>

                    </div>

                </div>

            </div>

        </div>

        <CreateCategoryModal ref="createModalRef" @create-category="handleCreateCategory" />

        <UpdateCategoryModal ref="updateModalRef" :slug="selectedSlug" @update-category="handleUpdateCategory" />

    </main>

</template>

<style scoped>

.fw-semibold {

    font-weight: 600;

}

.gap-2 {

    gap: 8px;

}

.img-thumbnail {

    border: 1px solid #ddd;

    padding: 2px;

}

.cursor-pointer {

    cursor: pointer;

}

</style>

1. Inside CreateCategoryModal.vue

<script setup>

import { ref, onMounted, onUnmounted } from 'vue';

import { Modal } from 'bootstrap';

import { useCategoryStore } from '../../../stores/category';

import Toastify from 'toastify-js'; // Import Toastify

const emit = defineEmits(['create-category']);

const modalRef = ref(null);

let bsModal = null;

const categoryStore = useCategoryStore();

const newCategory = ref({

    name: '',

    status: 'active',

    image: null,

});

const formErrors = ref({}); // Field-specific errors

// Initialize Bootstrap modal on mount

onMounted(() => {

    bsModal = new Modal(modalRef.value, {

        backdrop: true,

        keyboard: true,

    });

});

// Cleanup on unmount

onUnmounted(() => {

    if (bsModal) {

        bsModal.dispose();

    }

});

// Reset form and errors

const resetForm = () => {

    newCategory.value = {

        name: '',

        status: 'active',

        image: null,

    };

    formErrors.value = {};

    categoryStore.resetErrors();

    bsModal.hide();

};

// Handle image file selection

const handleImageChange = (event) => {

    newCategory.value.image = event.target.files[0];

};

// Handle form submission

const handleSubmit = async () => {

    formErrors.value = {};

    const formData = new FormData();

    formData.append('name', newCategory.value.name);

    formData.append('status', newCategory.value.status);

    if (newCategory.value.image) {

        formData.append('image', newCategory.value.image);

    }

    try {

        const createdCategory = await categoryStore.storeCategory(formData);

        // Emit event to parent (optional)

        emit('create-category', createdCategory);

        // Show success toast

        Toastify({

            text: 'Category created successfully!',

            duration: 3000, // 3 seconds

            close: true,

            gravity: 'bottom', // 'top' or 'bottom'

            position: 'right', // 'left', 'center', or 'right'

            backgroundColor: '#28a745', // Green for success

            stopOnFocus: true,

        }).showToast();

        // Close modal

        resetForm();

    } catch (error) {

        if (categoryStore.categoryErrors) {

            formErrors.value = categoryStore.categoryErrors;

        }

    }

};

defineExpose({

    open: () => {

        bsModal.show();

    },

});

</script>

<template>

    <div ref="modalRef" class="modal fade" tabindex="-1" role="dialog" aria-labelledby="createCategoryModalLabel"

        aria-hidden="true">

        <div class="modal-dialog modal-dialog-centered">

            <div class="modal-content">

                <div class="modal-header">

                    <h5 class="modal-title" id="createCategoryModalLabel">Create New Category</h5>

                    <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"

                        @click="resetForm"></button>

                </div>

                <div class="modal-body">

                    <form @submit.prevent="handleSubmit">

                        <!-- Name Field -->

                        <div class="mb-3">

                            <label for="categoryName" class="form-label">Category Name</label>

                            <input type="text" class="form-control" id="categoryName" v-model="newCategory.name"

                                placeholder="Enter category name" required :class="{ 'is-invalid': formErrors.name }">

                            <div v-if="formErrors.name" class="invalid-feedback">

                                {{ formErrors.name[0] }}

                            </div>

                        </div>

                        <!-- Status Field -->

                        <div class="mb-3">

                            <label for="categoryStatus" class="form-label">Status</label>

                            <select class="form-select" id="categoryStatus" v-model="newCategory.status"

                                :class="{ 'is-invalid': formErrors.status }">

                                <option value="active">Active</option>

                                <option value="inactive">Inactive</option>

                            </select>

                            <div v-if="formErrors.status" class="invalid-feedback">

                                {{ formErrors.status[0] }}

                            </div>

                        </div>

                        <!-- Image Field -->

                        <div class="mb-3">

                            <label for="categoryImage" class="form-label">Image</label>

                            <input type="file" class="form-control" id="categoryImage" accept="image/\*"

                                @change="handleImageChange" :class="{ 'is-invalid': formErrors.image }">

                            <div v-if="formErrors.image" class="invalid-feedback">

                                {{ formErrors.image[0] }}

                            </div>

                        </div>

                    </form>

                </div>

                <div class="modal-footer">

                    <button type="button" class="btn btn-secondary" data-bs-dismiss="modal" @click="resetForm">

                        Close

                    </button>

                    <button type="button" class="btn btn-primary" @click="handleSubmit"

                        :disabled="categoryStore.loading">

                        {{ categoryStore.loading ? 'Creating...' : 'Create Category' }}

                    </button>

                </div>

            </div>

        </div>

    </div>

</template>

<style scoped>

.modal-content {

    border-radius: 0.75rem;

}

.form-label {

    font-weight: 500;

}

.form-control,

.form-select {

    border-radius: 0.5rem;

}

@media (max-width: 576px) {

    .modal-dialog {

        margin: 1rem;

    }

}

</style>

1. Inside UpdateCategoryModal.vue

<script setup>

import { ref, onMounted, onUnmounted } from 'vue';

import { Modal } from 'bootstrap';

import { useCategoryStore } from '../../../stores/category';

import Toastify from 'toastify-js';

const props = defineProps({

    slug: {

        type: String,

        required: true,

    },

});

const emit = defineEmits(['update-category']);

const modalRef = ref(null);

let bsModal = null;

const categoryStore = useCategoryStore();

const categoryData = ref({

    name: '',

    status: 'active',

    image: null,

});

const formErrors = ref({});

// Initialize Bootstrap modal on mount

onMounted(() => {

    bsModal = new Modal(modalRef.value, {

        backdrop: true,

        keyboard: true,

    });

});

// Cleanup on unmount

onUnmounted(() => {

    if (bsModal) {

        bsModal.dispose();

    }

});

// Fetch category data and populate form

const fetchCategoryData = async (slug) => {

    try {

        await categoryStore.fetchCategory(slug);

        if (categoryStore.currentCategory) {

            categoryData.value = {

                name: categoryStore.currentCategory.name || '',

                status: categoryStore.currentCategory.status || 'active',

                image: null,

            };

        } else {

            throw new Error('Category not found');

        }

    } catch (error) {

        Toastify({

            text: 'Failed to load category data!',

            duration: 3000,

            close: true,

            gravity: 'bottom',

            position: 'right',

            backgroundColor: '#dc3545',

        }).showToast();

        resetForm();

    }

};

// Reset form and errors

const resetForm = () => {

    categoryData.value = {

        name: '',

        status: 'active',

        image: null,

    };

    formErrors.value = {};

    categoryStore.resetErrors();

    if (bsModal) {

        bsModal.hide();

    }

};

// Handle image file selection

const handleImageChange = (event) => {

    categoryData.value.image = event.target.files[0];

};

// Handle form submission

const handleSubmit = async () => {

    formErrors.value = {};

    const formData = new FormData();

    formData.append('name', categoryData.value.name);

    formData.append('status', categoryData.value.status);

    if (categoryData.value.image) {

        formData.append('image', categoryData.value.image);

    }

    try {

        const updatedCategory = await categoryStore.updateCategory(props.slug, formData);

        emit('update-category', updatedCategory);

        Toastify({

            text: 'Category updated successfully!',

            duration: 3000,

            close: true,

            gravity: 'bottom',

            position: 'right',

            backgroundColor: '#28a745',

        }).showToast();

        resetForm();

    } catch (error) {

        if (categoryStore.categoryErrors) {

            formErrors.value = categoryStore.categoryErrors;

        }

    }

};

// Open modal after fetching data, accepting slug as a parameter

const open = async (slug) => {

    await fetchCategoryData(slug); // Use the passed slug directly

    if (categoryStore.currentCategory) {

        bsModal.show();

    }

};

defineExpose({

    open,

});

</script>

<template>

    <div ref="modalRef" class="modal fade" tabindex="-1" role="dialog" aria-labelledby="updateCategoryModalLabel"

        aria-hidden="true">

        <div class="modal-dialog modal-dialog-centered">

            <div class="modal-content">

                <div class="modal-header">

                    <h5 class="modal-title" id="updateCategoryModalLabel">Update Category</h5>

                    <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"

                        @click="resetForm"></button>

                </div>

                <div class="modal-body">

                    <form @submit.prevent="handleSubmit">

                        <!-- Name Field -->

                        <div class="mb-3">

                            <label for="categoryName" class="form-label">Category Name</label>

                            <input type="text" class="form-control" id="categoryName" v-model="categoryData.name"

                                placeholder="Enter category name" required :class="{ 'is-invalid': formErrors.name }">

                            <div v-if="formErrors.name" class="invalid-feedback">

                                {{ formErrors.name[0] }}

                            </div>

                        </div>

                        <!-- Status Field -->

                        <div class="mb-3">

                            <label for="categoryStatus" class="form-label">Status</label>

                            <select class="form-select" id="categoryStatus" v-model="categoryData.status"

                                :class="{ 'is-invalid': formErrors.status }">

                                <option value="active">Active</option>

                                <option value="inactive">Inactive</option>

                            </select>

                            <div v-if="formErrors.status" class="invalid-feedback">

                                {{ formErrors.status[0] }}

                            </div>

                        </div>

                        <!-- Image Field -->

                        <div class="mb-3">

                            <label for="categoryImage" class="form-label">Image (optional)</label>

                            <input type="file" class="form-control" id="categoryImage" accept="image/\*"

                                @change="handleImageChange" :class="{ 'is-invalid': formErrors.image }">

                            <div v-if="formErrors.image" class="invalid-feedback">

                                {{ formErrors.image[0] }}

                            </div>

                        </div>

                    </form>

                </div>

                <div class="modal-footer">

                    <button type="button" class="btn btn-secondary" data-bs-dismiss="modal" @click="resetForm">

                        Close

                    </button>

                    <button type="button" class="btn btn-primary" @click="handleSubmit"

                        :disabled="categoryStore.loading">

                        {{ categoryStore.loading ? 'Updating...' : 'Update Category' }}

                    </button>

                </div>

            </div>

        </div>

    </div>

</template>

<style scoped>

.modal-content {

    border-radius: 0.75rem;

}

.form-label {

    font-weight: 500;

}

.form-control,

.form-select {

    border-radius: 0.5rem;

}

@media (max-width: 576px) {

    .modal-dialog {

        margin: 1rem;

    }

}

</style>