# **Authentication Using Laravel Sanctum (Token Based Authentication)**

## **Install necessary packages:**

composer require laravel/sanctum (if not already installed)

php artisan vendor:publish --provider="Laravel\Sanctum\SanctumServiceProvider"

php artisan migrate

## **Inside api.php**

<?php

use App\Http\Controllers\API\AuthController;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Route;

Route::post('/register', [AuthController::class, 'register']);

Route::post('/login', [AuthController::class, 'login']);

Route::post('/send-reset-password-email', [AuthController::class, 'sendPasswordResetLink']);

Route::post('/reset-password/{token}', [AuthController::class, 'resetPassword']);

Route::middleware(['auth:sanctum'])->group(function () {

    Route::post('/logout', [AuthController::class, 'logout']);

    Route::post('/change-password', [AuthController::class, 'changePassword']);

    Route::get('/user', [AuthController::class, 'user']);

});

## **Create API/AuthController:**

<?php

namespace App\Http\Controllers\API;

use App\Http\Controllers\Controller;

use App\Mail\PasswordResetMail;

use Illuminate\Http\Request;

use App\Models\User;

use Carbon\Carbon;

use Illuminate\Support\Facades\Hash;

use Illuminate\Support\Facades\Auth;

use Illuminate\Auth\Events\Registered;

use Illuminate\Support\Facades\Mail;

use Illuminate\Support\Facades\Password;

use Illuminate\Support\Str;

use Illuminate\Validation\ValidationException;

use Illuminate\Support\Facades\DB;

class AuthController extends Controller

{

    // Register

    public function register(Request $request)

    {

        try {

            $validatedData = $request->validate([

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

                'email' => 'required|string|email|max:255|unique:users',

                'password' => 'required|string|min:8|confirmed',

            ]);

            $user = User::create([

                'name' => $validatedData['name'],

                'email' => $validatedData['email'],

                'password' => Hash::make($validatedData['password']),

            ]);

            //verify email through email

            event(new Registered($user));

            $token = $user->createToken('auth\_token')->plainTextToken;

            return response()->json([

                'message' => 'User registered successfully. Please verify your email.',

                'access\_token' => $token,

                'token\_type' => 'Bearer',

            ], 201);

        } catch (ValidationException $e) {

            return response()->json([

                'message' => 'Validation failed',

                'errors' => $e->errors(),

            ], 422);

        }

    }

    // Login

    public function login(Request $request)

    {

        try {

            $validatedData = $request->validate([

                'email' => 'required|string|email',

                'password' => 'required|string',

            ]);

            $user = User::where('email', $validatedData['email'])->first();

            if (!$user || !Hash::check($validatedData['password'], $user->password)) {

                throw ValidationException::withMessages([

                    'email' => ['The provided credentials are incorrect.'],

                ]);

            }

            // verify email before email

            if (!$user->hasVerifiedEmail()) {

                return response()->json([

                    'message' => 'Please verify your email first'

                ], 403);

            }

            $token = $user->createToken('auth\_token')->plainTextToken;

            return response()->json([

                'access\_token' => $token,

                'token\_type' => 'Bearer',

                'user' => $user,

            ]);

        } catch (ValidationException $e) {

            return response()->json([

                'message' => 'Validation failed',

                'errors' => $e->errors(),

            ], 422);

        }

    }

    // Change Password

    public function changePassword(Request $request)

    {

        try {

            $validatedData = $request->validate([

                'current\_password' => 'required|string',

                'new\_password' => 'required|string|min:8|confirmed',

            ]);

            $user = Auth::user();

            if (!Hash::check($validatedData['current\_password'], $user->password)) {

                throw ValidationException::withMessages([

                    'current\_password' => ['Current password is incorrect'],

                ]);

            }

            $user->update([

                'password' => Hash::make($validatedData['new\_password'])

            ]);

            // Revoke all tokens

            $user->tokens()->delete();

            $token = $user->createToken('auth\_token')->plainTextToken;

            return response()->json([

                'message' => 'Password changed successfully',

                'access\_token' => $token,

                'token\_type' => 'Bearer',

            ]);

        } catch (ValidationException $e) {

            return response()->json([

                'message' => 'Validation failed',

                'errors' => $e->errors(),

            ], 422);

        }

    }

    // Logout

    public function logout(Request $request)

    {

        $request->user()->currentAccessToken()->delete();

        return response()->json([

            'message' => 'Logged out successfully'

        ]);

    }

    // Send Password Reset Link

    public function sendPasswordResetLink(Request $request)

    {

        try {

            // Validate the request

            $request->validate([

                'email' => 'required|email|exists:users,email',

            ]);

            $email = $request->email;

            // Generate a secure token

            $token = Str::random(60);

            // Save token to password\_resets table

            DB::table('password\_reset\_tokens')->updateOrInsert(

                ['email' => $email],

                [

                    'email' => $email,

                    'token' => $token, // Store raw token, not hashed

                    'created\_at' => now(),

                ]

            );

            // Send the email using a Mailable class

            Mail::to($email)->send(new PasswordResetMail($token, $email));

            return response()->json([

                'message' => 'Password reset link sent to your email',

                'status' => 'success',

            ], 200);

        } catch (ValidationException $e) {

            return response()->json([

                'message' => 'Validation failed',

                'errors' => $e->errors(),

                'status' => 'failed',

            ], 422);

        } catch (\Exception $e) {

            return response()->json([

                'message' => 'Failed to send password reset link',

                'error' => $e->getMessage(), // In production, you might want to hide detailed errors

                'status' => 'failed',

            ], 500);

        }

    }

    // Reset Password

    public function resetPassword(Request $request, $token)

    {

        try {

            // Clean up expired tokens (2 minutes expiration)

            DB::table('password\_reset\_tokens')

                ->where('created\_at', '<=', Carbon::now()->subMinutes(2))

                ->delete();

            // Validate the request

            $request->validate([

                'password' => 'required|confirmed|min:8', // Added min length for security

            ]);

            // Find the reset token

            $passwordReset = DB::table('password\_reset\_tokens')

                ->where('token', $token)

                ->first();

            if (!$passwordReset) {

                return response()->json([

                    'message' => 'Token is invalid or expired',

                    'status' => 'failed'

                ], 404);

            }

            // Update user's password

            $user = User::where('email', $passwordReset->email)->first();

            if (!$user) {

                return response()->json([

                    'message' => 'User not found',

                    'status' => 'failed'

                ], 404);

            }

            $user->password = Hash::make($request->password);

            $user->save();

            // Delete the used token

            DB::table('password\_reset\_tokens')

                ->where('email', $user->email)

                ->delete();

            return response()->json([

                'message' => 'Password reset successfully',

                'status' => 'success'

            ], 200);

        } catch (ValidationException $e) {

            return response()->json([

                'message' => 'Validation failed',

                'errors' => $e->errors(),

                'status' => 'failed'

            ], 422);

        } catch (\Exception $e) {

            return response()->json([

                'message' => 'Failed to reset password',

                'status' => 'failed'

            ], 500);

        }

    }

    // Get authenticated user

    public function user(Request $request)

    {

        return response()->json($request->user());

    }

}

## **Php artisan make:mail PasswordResetMail**

<?php

namespace App\Mail;

use Illuminate\Bus\Queueable;

use Illuminate\Mail\Mailable;

use Illuminate\Queue\SerializesModels;

class PasswordResetMail extends Mailable

{

    use Queueable, SerializesModels;

    public $token;

    public $email;

    /\*\*

     \* Create a new message instance.

     \*

     \* @param string $token

     \* @param string $email

     \* @return void

     \*/

    public function \_\_construct($token, $email)

    {

        $this->token = $token;

        $this->email = $email;

    }

    /\*\*

     \* Build the message.

     \*

     \* @return $this

     \*/

    public function build()

    {

        $frontendUrl = env('FRONTEND\_URL', 'http://localhost:3000'); // Fallback to localhost:3000 if not set

        return $this->subject('Reset Your Password')

            ->view('emails.reset')

            ->with([

                'token' => $this->token,

                'email' => $this->email,

                'frontendUrl' => $frontendUrl,

            ]);

    }

}

## **Inside .env**

MAIL\_MAILER=smtp

MAIL\_HOST=smtp.gmail.com

MAIL\_PORT=465

MAIL\_USERNAME=freelencerunittesting@gmail.com

MAIL\_PASSWORD=aqcdacybfhqsyvsc

MAIL\_ENCRYPTION=tls

MAIL\_FROM\_ADDRESS="freelencerunittesting@gmail.com"

MAIL\_FROM\_NAME="${APP\_NAME}"

## **Inside views/emails/reset.blade.php**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Reset Your Password</title>

    <style>

        body {

            font-family: Arial, sans-serif;

            margin: 0;

            padding: 0;

            background-color: #f4f4f4;

        }

        .container {

            max-width: 600px;

            margin: 50px auto;

            background-color: #ffffff;

            padding: 20px;

            border-radius: 8px;

            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

        }

        h1 {

            color: #333333;

            font-size: 24px;

        }

        p {

            color: #666666;

            line-height: 1.6;

        }

        .button {

            display: inline-block;

            padding: 12px 24px;

            background-color: #007bff;

            color: #ffffff;

            text-decoration: none;

            border-radius: 4px;

            font-weight: bold;

        }

        .button:hover {

            background-color: #0056b3;

        }

        hr {

            border: 0;

            border-top: 1px solid #eeeeee;

            margin: 20px 0;

        }

        .footer {

            margin-top: 20px;

            font-size: 12px;

            color: #999999;

            text-align: center;

        }

    </style>

</head>

<body>

    <div class="container">

        <h1>Reset Your Password</h1>

        <hr>

        <p>Hello,</p>

        <p>We received a request to reset your password. Click the button below to reset it. This link is valid for 60

            minutes.</p>

        <p>

            <a href="{{ $frontendUrl }}/reset-password?token={{ $token }}&email={{ urlencode($email) }}"

                class="button">

                Reset Password

            </a>

        </p>

        <p>If you did not request a password reset, please ignore this email or contact support if you have concerns.

        </p>

        <div class="footer">

            <p>&copy; {{ date('Y') }} {{ env('APP\_NAME') }}. All rights reserved.</p>

        </div>

    </div>

</body>

</html>

## **Inside .env**

FRONTEND\_URL=http://localhost:3000

SESSION\_DOMAIN=localhost

SANCTUM\_STATEFUL\_DOMAINS=localhost:3000

# **Setup VueJs with Vuetify:**

## **Install below necessary packages:**



## **Inside vite.config.js**

// inside vite.config.js

import { defineConfig } from 'vite'

import vue from '@vitejs/plugin-vue'

import path from 'path'

// https://vite.dev/config/

export default defineConfig({

  plugins: [vue()],

  server: {

    port: 3000, // Sets the development server to run on port 3000

  },

  resolve: {

    alias: {

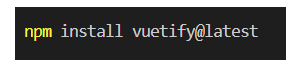
      '@': path.resolve(\_\_dirname, './src') // '@' will point to the src directory

    }

  }

})

## **Setup Vuetify:**



Inside src/plugins/vuetify.js

// src/plugins/vuetify.js

import { createVuetify } from 'vuetify';

import 'vuetify/styles'; // Import Vuetify styles

// Optional: Import components and directives if you want to use them explicitly

import \* as components from 'vuetify/components';

import \* as directives from 'vuetify/directives';

const vuetify = createVuetify({

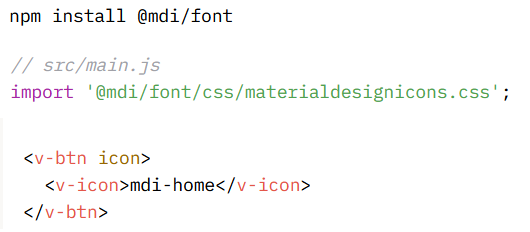
    components,

    directives,

});

export default vuetify

Now to use mdi icon:



Now, you can use icon and vuetify.

## **Default routing in src/router/index.js (Basic Routing)**

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

const routes = [

    {

        path: "/",

        name: "Frontend",

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

        children: [

            {

                path: "",

                name: "Home",

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

            },

        ],

    },

    {

        path: "/auth",

        name: "Auth",

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

        children: [

            {

                path: "/login",

                name: "Login",

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

            },

        ],

    },

    {

        path: "/admin",

        name: "Admin",

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

        children: [

            {

                path: "home",

                name: "Dashboard",

                component: () => import("@/views/backend/Home.vue"),

            },

        ],

    },

];

const router = createRouter({

    history: createWebHistory(),

    routes,

});

export default router;

## **Inside src/main.js**

// src/main.js

// Import Vue's createApp function

import { createApp } from 'vue'

// Import the main App component

import App from './App.vue'

// Import global Axios configuration

import './axios'

// Import Pinia (State Management)

import { createPinia } from 'pinia'

// Import Vue Router (Routing)

import router from './router'

// Import Vuetify configuration

import vuetify from './plugins/vuetify'

import '@mdi/font/css/materialdesignicons.css';

// Create an instance of Pinia

const pinia = createPinia();

// Create a Vue application instance

const app = createApp(App);

// Use Pinia for state management

app.use(pinia);

// Use Vue Router for handling routes

app.use(router);

// Use Vuetify for UI components

app.use(vuetify);

// Mount the app to the DOM

app.mount('#app');

## **Inside App.vue:**

<script setup>

import { ref } from 'vue';

import { useRouter } from 'vue-router';

const isLoading = ref(false);

const router = useRouter();

// Show loader when navigation starts

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

    isLoading.value = true;

    next();

});

// Hide loader when navigation is done

router.afterEach(() => {

    setTimeout(() => {

        isLoading.value = false;

    }, 500); // Small delay for smoother transition

});

</script>

<template>

    <main>

        <!-- Page Loader -->

        <v-overlay v-model="isLoading" class="loader-overlay" :scrim="false">

            <v-progress-circular indeterminate size="64" color="primary" />

        </v-overlay>

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

        <router-view />

    </main>

</template>

<style scoped>

/\* Page Loader Styles \*/

.loader-overlay {

    display: flex;

    align-items: center;

    justify-content: center;

    z-index: 9999;

    background: rgba(255, 255, 255, 0.8);

}

</style>

## **Inside FrontendLayout.vue ,BackendLayout.vue and AuthLayout.vue:**

<script setup>

</script>

<template>

    <main>

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

        <router-view />

    </main>

</template>

# **Login(without verification email) and Logout in vuejs:**

## **In axios.js**

// src/axios.js

import axios from 'axios'

const instance = axios.create({

    baseURL: 'http://localhost:8000/api',

    headers: {

        Accept: 'application/json',

    },

})

instance.interceptors.request.use((config) => {

    const token = localStorage.getItem('token')

    if (token) {

        config.headers.Authorization = `Bearer ${token}`

    }

    return config

})

instance.interceptors.response.use(

    (response) => response,

    (error) => {

        if (error.response && error.response.status === 401) {

            localStorage.removeItem('token')

            window.location.href = '/login'

        }

        return Promise.reject(error)

    }

)

export default instance

## **Inside stores/auth.js**

// src/stores/auth.js

import { ref, computed } from 'vue'

import { defineStore } from 'pinia'

import axios from '@/axios'

import router from '@/router'

/\*\*

 \* Authentication store to manage login, logout, token, and user state.

 \*/

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

    // Reactive state

    const user = ref(null)

    const token = ref(localStorage.getItem('token') || null)

    // Computed property to check if user is authenticated

    const isAuthenticated = computed(() => !!token.value)

    /\*\*

     \* Handles user login.

     \* @param {Object} credentials - User login credentials (email, password).

     \*/

    const login = async (credentials) => {

        try {

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

            // Store token in state and localStorage

            token.value = response.data.access\_token

            localStorage.setItem('token', token.value)

            // Fetch authenticated user data

            await fetchUser()

            // Redirect to admin dashboard after successful login

            router.push('/admin/home')

        } catch (error) {

            // Throw the backend validation or error message to the caller

            throw error.response?.data || { message: 'Login failed. Please try again.' }

        }

    }

    /\*\*

     \* Fetches the authenticated user data from backend.

     \*/

    const fetchUser = async () => {

        try {

            const response = await axios.get('/user', {

                headers: {

                    Authorization: `Bearer ${token.value}`,

                },

            })

            user.value = response.data

        } catch (error) {

            // If token is invalid or expired, force logout

            logout()

        }

    }

    /\*\*

     \* Logs the user out, clears local state and storage.

     \*/

    const logout = async () => {

        try {

            await axios.post('/logout', {}, {

                headers: {

                    Authorization: `Bearer ${token.value}`,

                },

            })

        } catch (error) {

            // Even if logout request fails, proceed with clearing state

            console.warn('Logout request failed, but continuing.')

        } finally {

            // Clear state and redirect to login page

            token.value = null

            user.value = null

            localStorage.removeItem('token')

            router.push('/login')

        }

    }

    return {

        user,

        token,

        isAuthenticated,

        login,

        logout,

        fetchUser,

    }

})

## **Inside index.js**

// src/router/index.js

import { createRouter, createWebHistory } from 'vue-router'

import { useAuthStore } from '@/stores/auth'

// Routes

const routes = [

    {

        path: '/',

        name: 'Frontend',

        component: () => import('@/layouts/FrontendLayout.vue'),

        children: [

            {

                path: '',

                name: 'Home',

                component: () => import('@/views/frontend/Home.vue'),

            },

        ],

    },

    {

        path: '/auth',

        name: 'Auth',

        component: () => import('@/layouts/AuthLayout.vue'),

        children: [

            {

                path: '/login',

                name: 'Login',

                component: () => import('@/views/auth/Login.vue'),

            },

        ],

    },

    {

        path: '/admin',

        name: 'Admin',

        component: () => import('@/layouts/BackendLayout.vue'),

        meta: { requiresAuth: true }, // 👈 Protected route

        children: [

            {

                path: 'home',

                name: 'Dashboard',

                component: () => import('@/views/backend/Home.vue'),

            },

        ],

    },

]

// Create router

const router = createRouter({

    history: createWebHistory(),

    routes,

})

// Global navigation guard

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

    const auth = useAuthStore()

    // Check if the route needs auth

    if (to.meta.requiresAuth && !auth.isAuthenticated) {

        return next('/login')

    }

    // Prevent access to login page if already authenticated

    if (to.path === '/login' && auth.isAuthenticated) {

        return next('/admin/home')

    }

    return next()

})

export default router

## **Inside Login.vue**

<!-- src/views/auth/Login.vue -->

<script setup>

import { ref } from 'vue'

import { useAuthStore } from '@/stores/auth'

const auth = useAuthStore()

const form = ref({

    email: '',

    password: '',

})

const loading = ref(false)

const errors = ref({

    email: null,

    password: null,

    general: null,

})

const handleLogin = async () => {

    errors.value = { email: null, password: null, general: null }

    loading.value = true

    try {

        await auth.login(form.value)

    } catch (err) {

        const e = err.errors || {}

        errors.value.email = e.email?.[0] || null

        errors.value.password = e.password?.[0] || null

        errors.value.general = err.message || 'Login failed'

    } finally {

        loading.value = false

    }

}

</script>

<template>

    <v-container class="d-flex align-center justify-center" style="height: 100vh">

        <v-card class="pa-6" min-width="400">

            <v-card-title class="text-h5">Login</v-card-title>

            <v-form @submit.prevent="handleLogin" ref="formRef">

                <v-text-field v-model="form.email" label="Email" :error-messages="errors.email" type="email" required />

                <v-text-field v-model="form.password" label="Password" :error-messages="errors.password" type="password"

                    required />

                <v-btn type="submit" color="primary" :loading="loading" block>Login</v-btn>

                <v-alert v-if="errors.general" type="error" class="mt-4" dense>

                    {{ errors.general }}

                </v-alert>

            </v-form>

        </v-card>

    </v-container>

</template>

## **Logout:**

<script setup>

import { useAuthStore } from "@/stores/auth"

// Access the authentication store

const authStore = useAuthStore();

// Methods

const logout = () => {

    authStore.logout();

};

</script>

<template>

    <v-btn class="mx-1" @click="logout" :loading="authStore.loading">

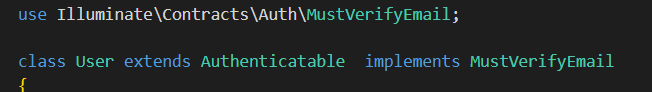
        Logout

    </v-btn>

</template>

# **Login(with email verification):**

## **Inside user model**



## **Inside AuthController:**

<?php

namespace App\Http\Controllers\API;

use App\Http\Controllers\Controller;

use App\Mail\PasswordResetMail;

use Illuminate\Http\Request;

use App\Models\User;

use Carbon\Carbon;

use Illuminate\Support\Facades\Hash;

use Illuminate\Support\Facades\Auth;

use Illuminate\Support\Facades\Mail;

use Illuminate\Support\Str;

use Illuminate\Validation\ValidationException;

use Illuminate\Support\Facades\DB;

use App\Mail\EmailVerificationMail;

class AuthController extends Controller

{

    // Register

     public function register(Request $request)

    {

        try {

            // Validate incoming request data

            $validatedData = $request->validate([

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

                'email' => 'required|string|email|max:255|unique:users',

                'password' => 'required|string|min:8|confirmed',

            ]);

            // Create new user

            $user = User::create([

                'name' => $validatedData['name'],

                'email' => $validatedData['email'],

                'password' => Hash::make($validatedData['password']),

                'verification\_token' => Str::random(60), // Always generate token for new user

            ]);

            // Send email verification link (always for new users)

            Mail::to($user->email)->send(new EmailVerificationMail($user));

            // Generate auth token

            $token = $user->createToken('auth\_token')->plainTextToken;

            // Return success response

            return response()->json([

                'message' => 'User registered successfully. Please verify your email.',

                'access\_token' => $token,

                'token\_type' => 'Bearer',

            ], 201);

        } catch (ValidationException $e) {

            // Return validation errors

            return response()->json([

                'message' => 'Validation failed',

                'errors' => $e->errors(),

            ], 422);

        }

    }

    // Login

    public function login(Request $request)

    {

        try {

            $validatedData = $request->validate([

                'email' => 'required|string|email',

                'password' => 'required|string',

            ]);

            $user = User::where('email', $validatedData['email'])->first();

            if (!$user || !Hash::check($validatedData['password'], $user->password)) {

                throw ValidationException::withMessages([

                    'email' => ['The provided credentials are incorrect.'],

                ]);

            }

            // Check if email is verified

            if (!$user->hasVerifiedEmail()) {

                // Generate new token if needed and send verification email

                if (!$user->verification\_token) {

                    $user->verification\_token = Str::random(60);

                    $user->save();

                }

                Mail::to($user->email)->send(new EmailVerificationMail($user));

                return response()->json([

                    'message' => 'email\_not\_verified',

                ], 403);

            }

            $token = $user->createToken('auth\_token')->plainTextToken;

            return response()->json([

                'access\_token' => $token,

                'token\_type' => 'Bearer',

                'user' => $user,

            ]);

        } catch (ValidationException $e) {

            return response()->json([

                'message' => 'Validation failed',

                'errors' => $e->errors(),

            ], 422);

        }

    }

    // Verify Email

    public function verifyEmail($token)

    {

        try {

            $user = User::where('verification\_token', $token)->first();

            if (!$user) {

                return response()->json([

                    'message' => 'Verification link has expired or is invalid.',

                    'status' => 'expired',

                ], 410); // 410 Gone for expired resources

            }

            if ($user->hasVerifiedEmail()) {

                return response()->json([

                    'message' => 'Email already verified.',

                    'status' => 'already\_verified',

                ], 200);

            }

            $user->email\_verified\_at = now();

            $user->verification\_token = null; // Clear token after use

            $user->save();

            $authToken = $user->createToken('auth\_token')->plainTextToken;

            return response()->json([

                'message' => 'Email verified successfully.',

                'access\_token' => $authToken,

                'token\_type' => 'Bearer',

                'status' => 'success',

            ], 200);

        } catch (\Exception $e) {

            return response()->json([

                'message' => 'Failed to verify email.',

                'error' => $e->getMessage(),

                'status' => 'failed',

            ], 500);

        }

    }

    // Resend Verification Email

    public function resendVerificationEmail(Request $request)

    {

        try {

            $request->validate([

                'email' => 'required|email|exists:users,email',

            ]);

            $user = User::where('email', $request->email)->first();

            if ($user->hasVerifiedEmail()) {

                return response()->json([

                    'message' => 'Email is already verified.',

                    'status' => 'already\_verified',

                ], 200);

            }

            // Generate new token

            $user->verification\_token = Str::random(60);

            $user->save();

            Mail::to($user->email)->send(new EmailVerificationMail($user));

            return response()->json([

                'message' => 'A fresh verification link has been sent to your email.',

                'status' => 'success',

            ], 200);

        } catch (ValidationException $e) {

            return response()->json([

                'message' => 'Validation failed',

                'errors' => $e->errors(),

                'status' => 'failed',

            ], 422);

        } catch (\Exception $e) {

            return response()->json([

                'message' => 'Failed to send verification email.',

                'error' => $e->getMessage(),

                'status' => 'failed',

            ], 500);

        }

    }

    // Change Password

    public function changePassword(Request $request)

    {

        try {

            $validatedData = $request->validate([

                'current\_password' => 'required|string',

                'new\_password' => 'required|string|min:8|confirmed',

            ]);

            $user = Auth::user();

            if (!Hash::check($validatedData['current\_password'], $user->password)) {

                throw ValidationException::withMessages([

                    'current\_password' => ['Current password is incorrect'],

                ]);

            }

            $user->update([

                'password' => Hash::make($validatedData['new\_password'])

            ]);

            // Revoke all tokens

            $user->tokens()->delete();

            $token = $user->createToken('auth\_token')->plainTextToken;

            return response()->json([

                'message' => 'Password changed successfully',

                'access\_token' => $token,

                'token\_type' => 'Bearer',

            ]);

        } catch (ValidationException $e) {

            return response()->json([

                'message' => 'Validation failed',

                'errors' => $e->errors(),

            ], 422);

        }

    }

    // Logout

    public function logout(Request $request)

    {

        $request->user()->currentAccessToken()->delete();

        return response()->json([

            'message' => 'Logged out successfully'

        ]);

    }

    // Send Password Reset Link

    public function sendPasswordResetLink(Request $request)

    {

        try {

            // Validate the request

            $request->validate([

                'email' => 'required|email|exists:users,email',

            ]);

            $email = $request->email;

            // Generate a secure token

            $token = Str::random(60);

            // Save token to password\_resets table

            DB::table('password\_reset\_tokens')->updateOrInsert(

                ['email' => $email],

                [

                    'email' => $email,

                    'token' => $token, // Store raw token, not hashed

                    'created\_at' => now(),

                ]

            );

            // Send the email using a Mailable class

            Mail::to($email)->send(new PasswordResetMail($token, $email));

            return response()->json([

                'message' => 'Password reset link sent to your email',

                'status' => 'success',

            ], 200);

        } catch (ValidationException $e) {

            return response()->json([

                'message' => 'Validation failed',

                'errors' => $e->errors(),

                'status' => 'failed',

            ], 422);

        } catch (\Exception $e) {

            return response()->json([

                'message' => 'Failed to send password reset link',

                'error' => $e->getMessage(), // In production, you might want to hide detailed errors

                'status' => 'failed',

            ], 500);

        }

    }

    // Reset Password

    public function resetPassword(Request $request, $token)

    {

        try {

            // Clean up expired tokens (2 minutes expiration)

            DB::table('password\_reset\_tokens')

                ->where('created\_at', '<=', Carbon::now()->subMinutes(2))

                ->delete();

            // Validate the request

            $request->validate([

                'password' => 'required|confirmed|min:8', // Added min length for security

            ]);

            // Find the reset token

            $passwordReset = DB::table('password\_reset\_tokens')

                ->where('token', $token)

                ->first();

            if (!$passwordReset) {

                return response()->json([

                    'message' => 'Token is invalid or expired',

                    'status' => 'failed'

                ], 404);

            }

            // Update user's password

            $user = User::where('email', $passwordReset->email)->first();

            if (!$user) {

                return response()->json([

                    'message' => 'User not found',

                    'status' => 'failed'

                ], 404);

            }

            $user->password = Hash::make($request->password);

            $user->save();

            // Delete the used token

            DB::table('password\_reset\_tokens')

                ->where('email', $user->email)

                ->delete();

            return response()->json([

                'message' => 'Password reset successfully',

                'status' => 'success'

            ], 200);

        } catch (ValidationException $e) {

            return response()->json([

                'message' => 'Validation failed',

                'errors' => $e->errors(),

                'status' => 'failed'

            ], 422);

        } catch (\Exception $e) {

            return response()->json([

                'message' => 'Failed to reset password',

                'status' => 'failed'

            ], 500);

        }

    }

    // Get authenticated user

    public function user(Request $request)

    {

        return response()->json($request->user());

    }

}

## **Make mail EmailVerificationMail.php:**

<?php

namespace App\Mail;

use App\Models\User;

use Illuminate\Bus\Queueable;

use Illuminate\Mail\Mailable;

use Illuminate\Queue\SerializesModels;

class EmailVerificationMail extends Mailable

{

    use Queueable, SerializesModels;

    public $user;

    public function \_\_construct(User $user)

    {

        $this->user = $user;

    }

    public function build()

    {

        return $this->subject('Verify Your Email Address')

            ->view('emails.verify-email')

            ->with([

                'verificationUrl' => "http://localhost:3000/verify-email/{$this->user->verification\_token}",

            ]);

    }

}

## **Inside views/emails/verify-email.blade.php**

<!DOCTYPE html>

<html>

<head>

    <title>Verify Your Email</title>

</head>

<body>

    <h1>Verify Your Email Address</h1>

    <p>Please click the link below to verify your email address:</p>

    <a href="{{ $verificationUrl }}">Verify Email</a>

    <p>If you did not create an account, no further action is required.</p>

</body>

</html>

## **Inside users table:**

 $table->string('verification\_token')->nullable();

## **Inside api.php**

<?php

use App\Http\Controllers\API\AuthController;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Route;

Route::post('/register', [AuthController::class, 'register']);

Route::post('/login', [AuthController::class, 'login']);

Route::post('/send-reset-password-email', [AuthController::class, 'sendPasswordResetLink']);

Route::post('/reset-password/{token}', [AuthController::class, 'resetPassword']);

// New routes for email verification

Route::get('/verify-email/{token}', [AuthController::class, 'verifyEmail']);

Route::post('/resend-verification-email', [AuthController::class, 'resendVerificationEmail']);

Route::middleware(['auth:sanctum'])->group(function () {

    Route::post('/logout', [AuthController::class, 'logout']);

    Route::post('/change-password', [AuthController::class, 'changePassword']);

    Route::get('/user', [AuthController::class, 'user']);

});

## **Inside auth.js**

// src/stores/auth.js

import { ref, computed } from 'vue'

import { defineStore } from 'pinia'

import axios from '@/axios'

import router from '@/router'

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

    const user = ref(null)

    const token = ref(localStorage.getItem('token') || null)

    const isAuthenticated = computed(() => !!token.value)

    // Login user and redirect

    const login = async (credentials) => {

        try {

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

            token.value = response.data.access\_token

            localStorage.setItem('token', token.value)

            await fetchUser()

            router.push('/admin/home')

        } catch (error) {

            // Handle unverified email

            if (error.response.data.message === 'email\_not\_verified') {

                router.push({ path: '/email-verification', query: { email: credentials.email } })

                return;

            }

            // Only throw errors for validation or server issues, not email verification

            if (error.response?.status !== 403) {

                throw error.response?.data || { message: 'Login failed. Please try again.' }

            }

        }

    }

    // Fetch user data

    const fetchUser = async () => {

        try {

            const response = await axios.get('/user')

            user.value = response.data

        } catch (error) {

            logout()

        }

    }

    // Logout user and clear state

    const logout = async () => {

        try {

            await axios.post('/logout')

        } catch (error) {

            console.warn('Logout request failed, but continuing.')

        } finally {

            token.value = null

            user.value = null

            localStorage.removeItem('token')

            router.push('/login')

        }

    }

    // Resend verification email

    const resendVerification = async (email) => {

        try {

            const response = await axios.post('/resend-verification-email', { email })

            return response.data

        } catch (error) {

            throw error.response?.data || { message: 'Failed to resend verification email.' }

        }

    }

    return {

        user,

        token,

        isAuthenticated,

        login,

        logout,

        fetchUser,

        resendVerification,

    }

})

## **Inside index.js**

// src/router/index.js

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

import { useAuthStore } from '@/stores/auth';

const routes = [

    {

        path: '/',

        name: 'Frontend',

        component: () => import('@/layouts/FrontendLayout.vue'),

        children: [

            {

                path: '',

                name: 'Home',

                component: () => import('@/views/frontend/Home.vue'),

            },

        ],

    },

    {

        path: '/auth',

        name: 'Auth',

        component: () => import('@/layouts/AuthLayout.vue'),

        children: [

            {

                path: '/login',

                name: 'Login',

                component: () => import('@/views/auth/Login.vue'),

            },

            {

                path: '/email-verification',

                name: 'EmailVerification',

                component: () => import('@/views/auth/EmailVerification.vue'),

            },

            {

                path: '/verify-email/:token',

                name: 'VerifyEmail',

                component: () => import('@/views/auth/VerifyEmail.vue'),

            },

            {

                path: '/register',

                name: 'Register',

                component: () => import('@/views/auth/Register.vue'),

            },

            {

                path: '/forgot-password',

                name: 'ForgotPassword',

                component: () => import('@/views/auth/ForgotPassword.vue'),

            },

            {

                path: '/reset-password',

                name: 'ResetPassword',

                component: () => import('@/views/auth/ResetPassword.vue'),

            },

        ],

    },

    {

        path: '/admin',

        name: 'Admin',

        component: () => import('@/layouts/BackendLayout.vue'),

        meta: { requiresAuth: true },

        children: [

            {

                path: 'home',

                name: 'Dashboard',

                component: () => import('@/views/backend/Home.vue'),

            },

            {

                path: 'category',

                name: 'Category',

                component: () => import('@/views/backend/category/Category.vue'),

            },

            {

                path: 'product',

                name: 'Product',

                component: () => import('@/views/backend/products/Product.vue'),

            },

            {

                path: 'product/add',

                name: 'AddProduct',

                component: () => import('@/views/backend/products/AddProduct.vue'),

            },

            {

                path: 'product/edit/:slug',

                name: 'EditProduct',

                component: () => import('@/views/backend/products/EditProduct.vue'),

            },

        ],

    },

];

const router = createRouter({

    history: createWebHistory(),

    routes,

});

// Navigation guard

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

    const auth = useAuthStore();

    // List of auth-related route names to restrict for authenticated users

    const authRoutes = [

        'Login',

        'EmailVerification',

        'VerifyEmail',

        'Register',

        'ForgotPassword',

        'ResetPassword',

    ];

    // If authenticated and trying to access an auth route, redirect to dashboard

    if (auth.isAuthenticated && authRoutes.includes(to.name)) {

        // Use next() with a path instead of returning to avoid overwriting history

        next('/admin/home');

    }

    // If unauthenticated and trying to access a protected route, redirect to login

    else if (to.meta.requiresAuth && !auth.isAuthenticated) {

        // Store the intended destination for post-login redirect (optional)

        sessionStorage.setItem('intendedRoute', to.fullPath);

        next('/login');

    }

    // Allow navigation to proceed normally

    else {

        next();

    }

});

export default router

## **Inside EmailVerification.vue**

<!-- src/views/auth/EmailVerification.vue -->

<script setup>

import { ref } from 'vue'

import { useAuthStore } from '@/stores/auth'

import { useRoute } from 'vue-router'

const auth = useAuthStore()

const route = useRoute()

const email = ref(route.query.email || '')

const loading = ref(false)

const message = ref('A fresh email verification link has been sent to you.')

const error = ref(null)

const resendVerification = async () => {

    if (!email.value) {

        error.value = 'Please provide your email address.'

        return

    }

    loading.value = true

    error.value = null

    try {

        const response = await auth.resendVerification(email.value)

        message.value = response.message

    } catch (err) {

        error.value = err.message || 'Failed to resend verification email.'

    } finally {

        loading.value = false

    }

}

</script>

<template>

    <v-container class="d-flex align-center justify-center" style="height: 100vh">

        <v-card class="pa-6" min-width="400">

            <v-card-title class="text-h5">Email Verification</v-card-title>

            <v-card-text>

                <p>{{ message }}</p>

                <v-text-field v-model="email" label="Email" type="email" :error-messages="error" required />

            </v-card-text>

            <v-card-actions>

                <v-btn color="primary" :loading="loading" block @click="resendVerification">

                    Resend Verification Link

                </v-btn>

            </v-card-actions>

            <v-alert v-if="error" type="error" class="mt-4" dense>

                {{ error }}

            </v-alert>

        </v-card>

    </v-container>

</template>

## **Inside VerifyEmail.vue**

<!-- src/views/auth/VerifyEmail.vue -->

<script setup>

import { ref, onMounted } from 'vue'

import { useRoute, useRouter } from 'vue-router'

import axios from '@/axios'

import { useAuthStore } from '@/stores/auth'

const route = useRoute()

const router = useRouter()

const auth = useAuthStore()

const loading = ref(true)

const message = ref('')

const error = ref(null)

const verifyEmail = async () => {

    try {

        const response = await axios.get(`/verify-email/${route.params.token}`)

        message.value = response.data.message

        if (response.data.status === 'success') {

            auth.token = response.data.access\_token

            localStorage.setItem('token', auth.token)

            await auth.fetchUser()

            setTimeout(() => router.push('/admin/home'), 2000)

        }

    } catch (err) {

        error.value = err.response?.data?.message || 'Failed to verify email.'

    } finally {

        loading.value = false

    }

}

onMounted(() => {

    verifyEmail()

})

</script>

<template>

    <v-container class="d-flex align-center justify-center" style="height: 100vh">

        <v-card class="pa-6" min-width="400">

            <v-card-title class="text-h5">Email Verification</v-card-title>

            <v-card-text>

                <v-progress-circular v-if="loading" indeterminate color="primary" class="mb-4" />

                <p v-if="message && !loading">{{ message }}</p>

                <v-alert v-if="error" type="error" dense>

                    {{ error }}

                </v-alert>

            </v-card-text>

        </v-card>

    </v-container>

</template>

## **Inside Login.vue**

<!-- src/views/auth/Login.vue -->

<script setup>

import { ref } from 'vue'

import { useAuthStore } from '@/stores/auth'

const auth = useAuthStore()

const form = ref({

    email: '',

    password: '',

})

const loading = ref(false)

const errors = ref({

    email: null,

    password: null,

    general: null,

})

const handleLogin = async () => {

    errors.value = { email: null, password: null, general: null }

    loading.value = true

    try {

        await auth.login(form.value)

    } catch (err) {

        const e = err.errors || {}

        errors.value.email = e.email?.[0] || null

        errors.value.password = e.password?.[0] || null

        errors.value.general = err.message || 'Login failed'

    } finally {

        loading.value = false

    }

}

</script>

<template>

    <v-container class="d-flex align-center justify-center" style="height: 100vh">

        <v-card class="pa-6" min-width="400">

            <v-card-title class="text-h5">Login</v-card-title>

            <v-form @submit.prevent="handleLogin" ref="formRef">

                <v-text-field v-model="form.email" label="Email" :error-messages="errors.email" type="email" required />

                <v-text-field v-model="form.password" label="Password" :error-messages="errors.password" type="password"

                    required />

                <v-btn type="submit" color="primary" :loading="loading" block>

                    Login

                </v-btn>

                <v-alert v-if="errors.general"

                    :type="errors.general === 'Please verify your email to login.' ? 'info' : 'error'" class="mt-4"

                    dense>

                    {{ errors.general }}

                </v-alert>

            </v-form>

        </v-card>

    </v-container>

</template>

# **Register User:**

## **Inside Register.vue**

<!-- src/views/auth/Register.vue -->

<script setup>

import { ref } from 'vue'

import { useAuthStore } from '@/stores/auth'

const auth = useAuthStore()

// Form data for registration

const form = ref({

    name: '',

    email: '',

    password: '',

    password\_confirmation: '',

})

const loading = ref(false)

const errors = ref({

    name: null,

    email: null,

    password: null,

    password\_confirmation: null,

    general: null,

})

// Handle registration submission

const handleRegister = async () => {

    errors.value = { name: null, email: null, password: null, password\_confirmation: null, general: null }

    loading.value = true

    try {

        await auth.register(form.value)

        // Redirect to EmailVerification.vue happens in auth.js

    } catch (err) {

        const e = err.errors || {}

        errors.value.name = e.name?.[0] || null

        errors.value.email = e.email?.[0] || null

        errors.value.password = e.password?.[0] || null

        errors.value.password\_confirmation = e.password\_confirmation?.[0] || null

        errors.value.general = err.message || 'Registration failed'

    } finally {

        loading.value = false

    }

}

</script>

<template>

    <v-container class="d-flex align-center justify-center" style="height: 100vh">

        <v-card class="pa-6" min-width="400">

            <v-card-title class="text-h5">Register</v-card-title>

            <v-form @submit.prevent="handleRegister" ref="formRef">

                <v-text-field v-model="form.name" label="Name" :error-messages="errors.name" required />

                <v-text-field v-model="form.email" label="Email" :error-messages="errors.email" type="email" required />

                <v-text-field v-model="form.password" label="Password" :error-messages="errors.password" type="password"

                    required />

                <v-text-field v-model="form.password\_confirmation" label="Confirm Password"

                    :error-messages="errors.password\_confirmation" type="password" required />

                <v-btn type="submit" color="primary" :loading="loading" block>

                    Register

                </v-btn>

                <v-alert v-if="errors.general" type="error" class="mt-4" dense>

                    {{ errors.general }}

                </v-alert>

            </v-form>

        </v-card>

    </v-container>

</template>

## **Inside auth.js**

 // Register new user and redirect to email verification

    const register = async (data) => {

        try {

            const response = await axios.post('/register', data)

            // Store token from registration

            token.value = response.data.access\_token

            localStorage.setItem('token', token.value)

            // Redirect to email verification page with email query

            router.push({ path: '/email-verification', query: { email: data.email } })

        } catch (error) {

            // Pass validation or server errors back to the component

            throw error.response?.data || { message: 'Registration failed. Please try again.' }

        }

    }

## **Inside index.js**

 {

                path: '/register',

                name: 'Register',

                component: () => import('@/views/auth/Register.vue'),

            },

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

    const auth = useAuthStore()

    // List of auth-related route names to restrict for authenticated users

    const authRoutes = ['Login', 'EmailVerification', 'VerifyEmail', 'Register']

    // Redirect authenticated users away from auth routes

    if (auth.isAuthenticated && authRoutes.includes(to.name)) {

        return next('/admin/home')

    }

    // Protect admin routes for unauthenticated users

    if (to.meta.requiresAuth && !auth.isAuthenticated) {

        return next('/login')

    }

    return next() // Proceed to the requested route

})

**Note: No need to change in backend logic.**

# **Reset Password:**

## **Auth.js**

// Send password reset link

    const sendPasswordResetLink = async (email) => {

        try {

            const response = await axios.post('/send-reset-password-email', { email })

            return response.data

        } catch (error) {

            throw error.response?.data || { message: 'Failed to send reset link' }

        }

    }

    // Reset password

    const resetPassword = async (token, data) => {

        try {

            const response = await axios.post(`/reset-password/${token}`, data)

            return response.data

        } catch (error) {

            throw error.response?.data || { message: 'Failed to reset password' }

        }

    }

## **ForgetPassword.vue**

<!-- src/views/auth/ForgotPassword.vue -->

<script setup>

import { ref, onUnmounted } from 'vue'

import { useAuthStore } from '@/stores/auth'

const auth = useAuthStore()

const email = ref('')

const loading = ref(false)

const message = ref('')

const error = ref(null)

const isButtonDisabled = ref(false) // Button starts enabled

const countdown = ref(0) // Countdown starts at 0 (no initial countdown)

let timer = null // Timer reference

// Start countdown timer after button click

const startCountdown = () => {

    isButtonDisabled.value = true

    countdown.value = 30

    timer = setInterval(() => {

        if (countdown.value > 0) {

            countdown.value--

        } else {

            clearInterval(timer)

            isButtonDisabled.value = false

        }

    }, 1000)

}

// Send reset link and start countdown

const sendResetLink = async () => {

    if (!email.value) {

        error.value = 'Email is required'

        return

    }

    loading.value = true

    message.value = ''

    error.value = null

    try {

        const response = await auth.sendPasswordResetLink(email.value)

        message.value = response.message // "Password reset link sent to your email"

        startCountdown() // Start countdown after successful send

    } catch (err) {

        error.value = err.message || 'Failed to send reset link'

    } finally {

        loading.value = false

    }

}

// Clean up timer on component unmount

onUnmounted(() => {

    if (timer) {

        clearInterval(timer)

    }

})

</script>

<template>

    <v-container fluid class="d-flex align-center justify-center"

        style="height: 100vh; background: linear-gradient(135deg, #e0eafc, #cfdef3);">

        <v-card class="pa-8 elevation-10" min-width="400" max-width="450" rounded="lg" color="white">

            <v-card-title class="text-h4 font-weight-bold primary--text text-center">

                Forgot Password

            </v-card-title>

            <v-card-subtitle class="text-center mt-2 grey--text">

                Enter your email to reset your password

            </v-card-subtitle>

            <v-form @submit.prevent="sendResetLink" class="mt-6">

                <v-text-field v-model="email" label="Email" type="email" prepend-inner-icon="mdi-email" outlined dense

                    color="primary" :error-messages="error" required />

                <v-btn type="submit" color="primary" :loading="loading" :disabled="isButtonDisabled" block large rounded

                    elevation="2" class="mt-6">

                    {{ isButtonDisabled ? `Send Reset Link (${countdown}s)` : 'Send Reset Link' }}

                </v-btn>

                <v-alert v-if="message" type="success" class="mt-6" dense outlined text>

                    {{ message }}

                </v-alert>

                <v-alert v-if="error" type="error" class="mt-6" dense outlined text>

                    {{ error }}

                </v-alert>

            </v-form>

            <router-link to="/login"

                class="text-decoration-none text-caption primary--text font-weight-medium mt-4 d-block text-center">

                Back to Login

            </router-link>

        </v-card>

    </v-container>

</template>

<style scoped>

a:hover {

    text-decoration: underline;

}

</style>

## **ResetPassword.vue**

<!-- src/views/auth/ResetPassword.vue -->

<script setup>

import { ref, onMounted } from 'vue'

import { useRoute, useRouter } from 'vue-router'

import { useAuthStore } from '@/stores/auth'

const route = useRoute()

const router = useRouter()

const auth = useAuthStore()

const form = ref({

    password: '',

    password\_confirmation: '',

})

const email = ref('')

const token = ref('')

const loading = ref(false)

const message = ref('')

const error = ref(null)

onMounted(() => {

    email.value = route.query.email || ''

    token.value = route.query.token || ''

    if (!token.value || !email.value) {

        error.value = 'Invalid reset link'

    }

})

const resetPassword = async () => {

    if (!token.value || !email.value) return

    loading.value = true

    message.value = ''

    error.value = null

    try {

        const response = await auth.resetPassword(token.value, form.value)

        message.value = response.message // "Password reset successfully"

        // Check email verification status

        await auth.fetchUser()

        if (auth.user?.email\_verified\_at) {

            setTimeout(() => router.push('/admin/home'), 2000) // Redirect after 2s

        } else {

            setTimeout(() => router.push({ path: '/auth/email-verification', query: { email: email.value } }), 2000)

        }

    } catch (err) {

        error.value = err.message || 'Failed to reset password'

    } finally {

        loading.value = false

    }

}

</script>

<template>

    <v-container fluid class="d-flex align-center justify-center"

        style="height: 100vh; background: linear-gradient(135deg, #e0eafc, #cfdef3);">

        <v-card class="pa-8 elevation-10" min-width="400" max-width="450" rounded="lg" color="white">

            <v-card-title class="text-h4 font-weight-bold primary--text text-center">

                Reset Password

            </v-card-title>

            <v-card-subtitle class="text-center mt-2 grey--text">

                Enter your new password

            </v-card-subtitle>

            <v-form @submit.prevent="resetPassword" class="mt-6">

                <v-text-field v-model="form.password" label="New Password" type="password" prepend-inner-icon="mdi-lock"

                    outlined dense color="primary"

                    :error-messages="error && !form.password ? 'Password is required' : null" required />

                <v-text-field v-model="form.password\_confirmation" label="Confirm Password" type="password"

                    prepend-inner-icon="mdi-lock-check" outlined dense color="primary"

                    :error-messages="error && !form.password\_confirmation ? 'Confirmation is required' : null"

                    required />

                <v-btn type="submit" color="primary" :loading="loading" block large rounded elevation="2" class="mt-6"

                    :disabled="!token || !email">

                    Reset Password

                </v-btn>

                <v-alert v-if="message" type="success" class="mt-6" dense outlined text>

                    {{ message }}

                </v-alert>

                <v-alert v-if="error" type="error" class="mt-6" dense outlined text>

                    {{ error }}

                </v-alert>

            </v-form>

            <router-link to="/auth/login"

                class="text-decoration-none text-caption primary--text font-weight-medium mt-4 d-block text-center">

                Back to Login

            </router-link>

        </v-card>

    </v-container>

</template>

<style scoped>

a:hover {

    text-decoration: underline;

}

</style>

## **Inside index.js**

{

                path: '/forgot-password',

                name: 'ForgotPassword',

                component: () => import('@/views/auth/ForgotPassword.vue'),

            },

            {

                path: '/reset-password',

                name: 'ResetPassword',

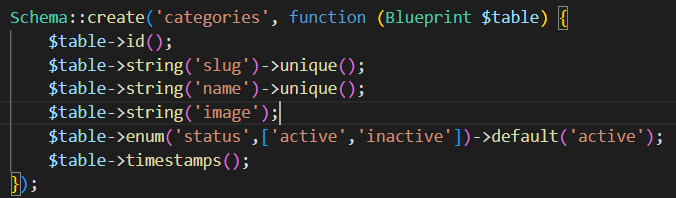
                component: () => import('@/views/auth/ResetPassword.vue'),

            },

**Note: No need to change in backend**

# **CRUD OPERATION:**

## **Make model and migration for category.**



<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

use Illuminate\Support\Str;

class Category extends Model

{

    use HasFactory;

    protected $fillable = ['slug', 'name', 'status', 'image'];

    // use slug instead of id

    public function getRouteKeyName()

    {

        return 'slug';

    }

    // boot

    protected static function boot()

    {

        parent::boot();

        static::creating(function ($category) {

            $category->slug = static::generateUniqueSlug();

        });

    }

    // generate new slug when category created

    private static function generateUniqueSlug()

    {

        do {

            $slug = Str::random(8); // Generates an 8-character random string

        } while (self::where('slug', $slug)->exists()); // Ensure uniqueness

        return $slug;

    }

}

## **Make controller: php artisan make:controller Api/CategoryController --api**

<?php

namespace App\Http\Controllers\Api;

use App\Http\Controllers\Controller;

use App\Http\Resources\CategoryResource;

use App\Models\Category;

use Exception;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\DB;

use Illuminate\Support\Facades\Storage;

use Illuminate\Support\Facades\Validator;

class CategoryController extends Controller

{

    /\*\*

     \* Display a listing of the resource.

     \*/

    public function index(Request $request)

    {

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

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

        $status = $request->query('status', ''); // New status filter

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

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

        $query = Category::query();

        if ($search) {

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

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

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

            });

        }

        if ($status) {

            $query->where('status', $status); // Exact match for status

        }

        $allowedSorts = ['name', 'status', 'updated\_at']; // Updated to include updated\_at

        $sortBy = in\_array($sortBy, $allowedSorts) ? $sortBy : 'updated\_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|unique:categories,name',

            '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'   => 'required|string|max:255|unique:categories,name,' . $category->id,

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

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

        ]);

        if ($validator->fails()) {

            return response()->json([

                'message' => 'Validation errors',

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

            ], 422);

        }

        $data = $request->only(['name', 'status']);

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

            // Delete old image if exists

            if ($category->image && Storage::disk('public')->exists($category->image)) {

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

            }

            // Store new image

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

        }

        $category->update($data);

        return (new CategoryResource($category))->response()->setStatusCode(200);

    }

    /\*\*

     \* Remove the specified resource from storage.

     \*/

    public function destroy(Category $category)

    {

        try {

            // Delete image if exists

            if ($category->image && Storage::disk('public')->exists($category->image)) {

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

            }

            // Delete category

            $category->delete();

            return response()->json([

                'message' => 'Category deleted successfully.'

            ], 200);

        } catch (Exception $e) {

            return response()->json([

                'message' => 'Failed to delete the category.',

                'error' => $e->getMessage(),

            ], 500);

        }

    }

    /\*\*

     \* Delete multiple categories

     \*/

    public function deleteMultiple(Request $request)

    {

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

            'slugs' => 'required|array',

            'slugs.\*' => 'required|string|exists:categories,slug',

        ]);

        if ($validator->fails()) {

            return response()->json([

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

            ], 422);

        }

        $slugs = $request->input('slugs');

        DB::transaction(function () use ($slugs) {

            $categories = Category::whereIn('slug', $slugs)->get();

            foreach ($categories as $category) {

                if ($category->image) {

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

                }

                $category->delete();

            }

        });

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

    }

}

## **Make CategoryResource:(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->diffForHumans(),

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

        ];

    }

## **Inside api.php**

// categories

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

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

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

        Route::delete('/multiple', [CategoryController::class, 'deleteMultiple'])->name('categories.deleteMultiple');

        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');

    });

Or



## **Inside stores/category.js(for single page)**

import { defineStore } from 'pinia';

import { ref } from 'vue';

import api from '@/axios';

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

    // ========= State =========

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

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

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

    const loading = ref(false); // Loading state

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

    const activeCategories = ref([]); // New state for active categories

    // ======= Fetch all categories (GET /api/category) =======

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

        try {

            loading.value = true;

            const params = {

                page,

                per\_page: perPage,

                sort\_by: sortBy,

                sort\_direction: sortDirection,

            };

            if (search) params.search = encodeURIComponent(search);

            if (status) params.status = encodeURIComponent(status);

            const response = await api.get('/category', { params });

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

            categories.value = response.data.data || [];

            pagination.value = {

                total: response.data.meta?.total || 0,

                current\_page: response.data.meta?.current\_page || 1,

                last\_page: response.data.meta?.last\_page || 1,

                per\_page: response.data.meta?.per\_page || perPage,

            };

            return response.data;

        } catch (error) {

            console.error('Fetch Categories Error:', error.response?.data || error.message);

            categories.value = [];

            pagination.value = { total: 0, current\_page: 1, last\_page: 1, per\_page: perPage };

            throw error;

        } finally {

            loading.value = false;

        }

    };

    // ======= Fetch a single category (GET /api/category/{slug}) =======

    const fetchCategory = async (slug) => {

        try {

            loading.value = true;

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

            currentCategory.value = response.data.data;

            return response;

        } catch (error) {

            console.error('Fetch Category Error:', error.response?.data || error.message);

            categoryErrors.value = error.response?.data?.errors || { general: ['Category not found'] };

            currentCategory.value = null;

            throw error;

        } finally {

            loading.value = false;

        }

    };

    // ======= Store a new category (POST /api/category) =======

    const storeCategory = async (formData) => {

        try {

            loading.value = true;

            categoryErrors.value = {};

            const response = await api.post('/category', formData, {

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

            });

            const newCategory = response.data.data;

            categories.value.unshift(newCategory);

            return newCategory;

        } catch (error) {

            console.error('Store Category Error:', error.response?.data || error.message);

            categoryErrors.value = error.response?.data?.errors || { general: ['Failed to create category'] };

            throw error;

        } finally {

            loading.value = false;

        }

    };

    // ======= Update an existing category (PUT /api/category/{slug}) =======

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

        try {

            loading.value = true;

            categoryErrors.value = {};

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

            const response = await api.post(`/category/${slug}`, formData, {

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

            });

            const updatedCategory = response.data.data;

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

            if (index !== -1) categories.value[index] = updatedCategory;

            currentCategory.value = updatedCategory;

            return updatedCategory;

        } catch (error) {

            console.error('Update Category Error:', error.response?.data || error.message);

            categoryErrors.value = error.response?.data?.errors || { general: ['Failed to update category'] };

            throw error;

        } finally {

            loading.value = false;

        }

    };

    // ======= Delete a category (DELETE /api/category/{slug}) =======

    const deleteCategory = async (slug) => {

        try {

            loading.value = true;

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

            categories.value = categories.value.filter(cat => cat.slug !== slug);

            if (currentCategory.value?.slug === slug) currentCategory.value = null;

            return slug;

        } catch (error) {

            console.error('Delete Category Error:', error.response?.data || error.message);

            categoryErrors.value = error.response?.data?.errors || { general: ['Failed to delete category'] };

            throw error;

        } finally {

            loading.value = false;

        }

    };

    // ======= Delete multiple categories (DELETE /api/category/multiple) =======

    const deleteMultipleCategories = async (slugs) => {

        try {

            loading.value = true;

            await api.delete('/category/multiple', { data: { slugs } });

            categories.value = categories.value.filter(cat => !slugs.includes(cat.slug));

            if (currentCategory.value && slugs.includes(currentCategory.value.slug)) {

                currentCategory.value = null;

            }

            return slugs;

        } catch (error) {

            console.error('Delete Multiple Categories Error:', error.response?.data || error.message);

            categoryErrors.value = error.response?.data?.errors || { general: ['Failed to delete categories'] };

            throw error;

        } finally {

            loading.value = false;

        }

    };

    // ======= Reset errors =======

    const resetErrors = () => {

        categoryErrors.value = {};

    };

    // Fetch active categories (GET /api/categories/active)

    const fetchActiveCategories = async () => {

        try {

            loading.value = true;

            const response = await api.get('/category/active');

            activeCategories.value = response.data.data || [];

        } catch (error) {

            console.error('Fetch Active Categories Error:', error.response?.data || error.message);

            activeCategories.value = [];

            throw error;

        } finally {

            loading.value = false;

        }

    };

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

    return {

        categories,

        currentCategory,

        categoryErrors,

        loading,

        pagination,

        fetchCategories,

        fetchCategory,

        storeCategory,

        updateCategory,

        deleteCategory,

        deleteMultipleCategories,

        resetErrors,

        fetchActiveCategories,

        activeCategories

    };

});

## **Inside Category.vue (CRUD in single page)**

<script setup>

import { ref, watch } from 'vue';

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

import debounce from 'lodash/debounce';

const categoryStore = useCategoryStore();

// Table state

const page = ref(1);

const itemsPerPage = ref(10);

const search = ref('');

const sortBy = ref([{ key: 'updated\_at', order: 'desc' }]);

const statusFilter = ref('');

const selectedCategories = ref([]);

// Image Modal state

const showImageModal = ref(false);

const selectedImage = ref(null);

// Add/Edit Modal state

const showAddModal = ref(false);

const showEditModal = ref(false);

const newCategory = ref({ slug: null, name: '', status: 'active', image: null });

const addForm = ref(null);

const serverErrors = ref({});

// Validation rules

const rules = {

    name: [

        v => !!v || 'Name is required',

        v => (v && v.length <= 255) || 'Name must be less than 255 characters',

    ],

    status: [

        v => !!v || 'Status is required',

        v => ['active', 'inactive'].includes(v) || 'Status must be "active" or "inactive"',

    ],

    image: [

        v => !v || v.size <= 2 \* 1024 \* 1024 || 'Image must be less than 2MB',

    ],

};

// Table headers

const headers = [

    { title: '', key: 'data-table-select', sortable: false, width: '50px' },

    { title: 'Image', key: 'image', sortable: false },

    { title: 'Name', key: 'name', sortable: true },

    { title: 'Status', key: 'status', sortable: true },

    { title: 'Created At', key: 'created\_at', sortable: true },

    { title: 'Actions', key: 'actions', sortable: false },

];

// Load items from server

const loadItems = debounce(async (options) => {

    page.value = options.page;

    itemsPerPage.value = options.itemsPerPage;

    const sort = options.sortBy.length ? options.sortBy[0] : { key: 'updated\_at', order: 'desc' };

    sortBy.value = [sort];

    await categoryStore.fetchCategories(

        page.value,

        itemsPerPage.value,

        search.value,

        sort.key,

        sort.order,

        statusFilter.value

    );

}, 300);

// Debounced filter update

const updateFilters = debounce(() => {

    page.value = 1;

    loadItems({ page: page.value, itemsPerPage: itemsPerPage.value, sortBy: sortBy.value });

}, 500);

// Watch filters

watch([search, statusFilter], updateFilters);

// Image modal handlers

const openImageModal = (imageUrl) => {

    selectedImage.value = imageUrl;

    showImageModal.value = true;

};

// Add/Edit modal handlers

const resetNewCategory = () => ({ slug: null, name: '', status: 'active', image: null });

const openAddModal = () => {

    newCategory.value = resetNewCategory();

    categoryStore.resetErrors();

    serverErrors.value = {};

    showAddModal.value = true;

};

const closeAddModal = () => {

    showAddModal.value = false;

};

const openEditModal = async (slug) => {

    try {

        await categoryStore.fetchCategory(slug);

        newCategory.value = {

            slug,

            name: categoryStore.currentCategory.name,

            status: categoryStore.currentCategory.status,

            image: null,

        };

        categoryStore.resetErrors();

        serverErrors.value = {};

        showEditModal.value = true;

    } catch (error) {

        console.error('Error opening edit modal:', error);

    }

};

const closeEditModal = () => {

    showEditModal.value = false;

    newCategory.value = resetNewCategory();

};

// Submit handlers

const submitCategory = async () => {

    const { valid } = await addForm.value.validate();

    if (!valid) return;

    try {

        await categoryStore.storeCategory(newCategory.value);

        showAddModal.value = false;

    } catch (error) {

        serverErrors.value = categoryStore.categoryErrors;

    }

};

const submitEditCategory = async () => {

    const { valid } = await addForm.value.validate();

    if (!valid) return;

    try {

        await categoryStore.updateCategory(newCategory.value.slug, newCategory.value);

        showEditModal.value = false;

    } catch (error) {

        serverErrors.value = categoryStore.categoryErrors;

    }

};

// Status chip color

const getStatusColor = status => (status === 'active' ? 'green' : 'red');

// CRUD operations

const editCategory = slug => openEditModal(slug);

const deleteCategory = async (slug) => {

    if (!confirm('Are you sure you want to delete this category?')) return;

    try {

        await categoryStore.deleteCategory(slug);

    } catch (error) {

        alert('Failed to delete category. Please try again.');

    }

};

const deleteMultipleCategories = async () => {

    if (!confirm('Are you sure you want to delete the selected categories?')) return;

    try {

        await categoryStore.deleteMultipleCategories(selectedCategories.value);

        selectedCategories.value = [];

    } catch (error) {

        alert('Failed to delete categories. Please try again.');

    }

};

</script>

<template>

    <v-container class="py-6">

        <!-- Filters and Search -->

        <v-card class="pa-4 mb-4 rounded-lg elevation-2">

            <v-row align="center" justify="space-between">

                <v-col cols="12" sm="6" md="4">

                    <v-text-field v-model="search" label="Search Categories" prepend-inner-icon="mdi-magnify"

                        variant="outlined" clearable density="comfortable" class="rounded-lg" />

                </v-col>

                <v-col cols="12" sm="6" md="4">

                    <v-select v-model="statusFilter" :items="[

                        { title: 'All', value: '' },

                        { title: 'Active', value: 'active' },

                        { title: 'Inactive', value: 'inactive' }

                    ]" label="Filter by Status" prepend-inner-icon="mdi-filter" variant="outlined"

                        density="comfortable" class="rounded-lg" clearable />

                </v-col>

                <v-spacer />

                <v-col cols="12" sm="auto">

                    <v-btn color="teal-darken-2" class="rounded-lg px-4 mr-2" @click="openAddModal">

                        <v-icon left>mdi-plus</v-icon> Add Category

                    </v-btn>

                    <v-btn color="red-darken-2" class="rounded-lg px-4" :disabled="selectedCategories.length === 0"

                        @click="deleteMultipleCategories">

                        <v-icon left>mdi-delete</v-icon> Delete Selected

                    </v-btn>

                </v-col>

            </v-row>

        </v-card>

        <!-- Data Table -->

        <v-card class="rounded-lg elevation-2">

            <v-data-table-server v-model:items-per-page="itemsPerPage" :headers="headers"

                :items="categoryStore.categories" :items-length="categoryStore.pagination.total || 0"

                :loading="categoryStore.loading" item-value="slug" v-model:sort-by="sortBy" v-model="selectedCategories"

                show-select :items-per-page-options="[5, 10, 20, 50]" @update:options="loadItems">

                <template v-slot:item.image="{ item }">

                    <v-img v-if="item.image" :src="item.image" height="40" width="40"

                        class="my-2 rounded-lg cursor-pointer" @click="openImageModal(item.image)" />

                    <span v-else>No Image</span>

                </template>

                <template v-slot:item.status="{ item }">

                    <v-chip :color="getStatusColor(item.status)" small class="text-uppercase font-weight-bold">

                        {{ item.status }}

                    </v-chip>

                </template>

                <template v-slot:item.actions="{ item }">

                    <v-btn icon color="warning" size="small" @click="editCategory(item.slug)">

                        <v-icon>mdi-pencil</v-icon>

                    </v-btn>

                    <v-btn icon color="red-darken-2" size="small" class="ms-2" @click="deleteCategory(item.slug)">

                        <v-icon>mdi-delete</v-icon>

                    </v-btn>

                </template>

            </v-data-table-server>

        </v-card>

        <!-- Image Modal -->

        <v-dialog v-model="showImageModal" max-width="500">

            <v-card class="rounded-lg elevation-3">

                <v-card-title class="d-flex align-center" style="background: linear-gradient(45deg, #00695c, #00897b)">

                    <span class="text-h6 font-weight-bold text-white">Category Image</span>

                    <v-spacer />

                    <v-btn icon @click="showImageModal = false" variant="plain">

                        <v-icon color="white">mdi-close</v-icon>

                    </v-btn>

                </v-card-title>

                <v-card-text class="text-center py-4">

                    <v-img :src="selectedImage" max-height="400" contain class="mx-auto rounded-lg" />

                </v-card-text>

            </v-card>

        </v-dialog>

        <!-- Add Category Modal -->

        <v-dialog v-model="showAddModal" max-width="500" persistent transition="dialog-bottom-transition">

            <v-card rounded="xl" elevation="12">

                <v-card-title class="py-4 px-6" style="background: linear-gradient(45deg, #00695c, #00897b)">

                    <v-row align="center">

                        <v-col>

                            <span class="text-h6 font-weight-medium text-white">Add New Category</span>

                        </v-col>

                        <v-col cols="auto">

                            <v-btn icon @click="closeAddModal" variant="text" color="white" class="mt-n1">

                                <v-icon>mdi-close</v-icon>

                            </v-btn>

                        </v-col>

                    </v-row>

                </v-card-title>

                <v-card-text class="pa-6">

                    <v-form ref="addForm" @submit.prevent="submitCategory" class="d-flex flex-column gap-4">

                        <v-text-field v-model="newCategory.name" label="Category Name" prepend-icon="mdi-tag"

                            density="comfortable" variant="outlined" color="teal-darken-2" class="rounded-lg mb-4"

                            :rules="rules.name" required :error-messages="serverErrors.name" />

                        <v-select v-model="newCategory.status" :items="[

                            { title: 'Active', value: 'active' },

                            { title: 'Inactive', value: 'inactive' }

                        ]" label="Status" prepend-icon="mdi-toggle-switch" density="comfortable" variant="outlined"

                            color="teal-darken-2" class="rounded-lg mb-4" :rules="rules.status" required

                            :error-messages="serverErrors.status" />

                        <v-file-input v-model="newCategory.image" label="Category Image" prepend-icon="mdi-image"

                            accept="image/\*" density="comfortable" variant="outlined" class="rounded-lg mb-4"

                            color="teal-darken-2" :rules="rules.image" required :error-messages="serverErrors.image" />

                        <v-btn type="submit" color="teal-darken-2" variant="flat" size="large" block

                            :loading="categoryStore.loading" class="mt-4 rounded-lg" elevation="2">

                            Save Category

                        </v-btn>

                    </v-form>

                </v-card-text>

            </v-card>

        </v-dialog>

        <!-- Edit Category Modal -->

        <v-dialog v-model="showEditModal" max-width="500" persistent transition="dialog-bottom-transition">

            <v-card rounded="xl" elevation="12">

                <v-card-title class="py-4 px-6" style="background: linear-gradient(45deg, #00695c, #00897b)">

                    <v-row align="center">

                        <v-col>

                            <span class="text-h6 font-weight-medium text-white">Edit Category</span>

                        </v-col>

                        <v-col cols="auto">

                            <v-btn icon @click="closeEditModal" variant="text" color="white" class="mt-n1">

                                <v-icon>mdi-close</v-icon>

                            </v-btn>

                        </v-col>

                    </v-row>

                </v-card-title>

                <v-card-text class="pa-6">

                    <v-form ref="addForm" @submit.prevent="submitEditCategory" class="d-flex flex-column gap-4">

                        <v-text-field v-model="newCategory.name" label="Category Name" prepend-icon="mdi-tag"

                            density="comfortable" variant="outlined" color="teal-darken-2" class="rounded-lg mb-4"

                            :rules="rules.name" required :error-messages="serverErrors.name" />

                        <v-select v-model="newCategory.status" :items="[

                            { title: 'Active', value: 'active' },

                            { title: 'Inactive', value: 'inactive' }

                        ]" label="Status" prepend-icon="mdi-toggle-switch" density="comfortable" variant="outlined"

                            color="teal-darken-2" class="rounded-lg mb-4" :rules="rules.status" required

                            :error-messages="serverErrors.status" />

                        <v-file-input v-model="newCategory.image" label="Category Image" prepend-icon="mdi-image"

                            accept="image/\*" density="comfortable" variant="outlined" class="rounded-lg mb-4"

                            color="teal-darken-2" :rules="rules.image" :error-messages="serverErrors.image" />

                        <v-btn type="submit" color="teal-darken-2" variant="flat" size="large" block

                            :loading="categoryStore.loading" class="mt-4 rounded-lg" elevation="2">

                            Update Category

                        </v-btn>

                    </v-form>

                </v-card-text>

            </v-card>

        </v-dialog>

    </v-container>

</template>

<style scoped>

.cursor-pointer {

    cursor: pointer;

}

.gap-4 {

    gap: 1rem;

}

</style>

## **Inside index.js**

{

                path: "category",

                name: "Category",

                component: () => import("@/views/backend/category/Category.vue"),

            },

## **How to use alert message in toastify?**



Inside main.js

import 'vue3-toastify/dist/index.css'; // Import default styles

Inside src/utils/toast.js

import { toast } from 'vue3-toastify';

export const showToast = {

    success(message) {

        toast.success(message, {

            icon: '✅',

        });

    },

    error(message) {

        toast.error(message, {

            icon: '❌',

        });

    },

    info(message) {

        toast.info(message, {

            icon: 'ℹ️',

        });

    },

    warning(message) {

        toast.warning(message, {

            icon: '⚠️',

        });

    },

};

Suppose in category.js

import { showToast } from '@/utils/toast'; // Import toast utility

  showToast.success('Product added successfully!'); // Use vue3-toastify

# **CRUD for Multiple page:**

## **Make controller: php artisan make:controller ProductController –api**

class ProductController extends Controller

{

    /\*\*

     \* Display a listing of the resource.

     \*/

    public function index(Request $request)

    {

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

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

        $status = $request->query('status', ''); // New status filter

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

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

        $query = Product::with('category');

        if ($search) {

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

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

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

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

            });

        }

        if ($status) {

            $query->where('status', $status); // Exact match for status

        }

        $allowedSorts = ['name', 'status', 'updated\_at']; // Updated to include updated\_at

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

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

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

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

        return ProductResource::collection($products);

    }

    /\*\*

     \* Store a newly created resource in storage.

     \*/

    public function store(Request $request)

    {

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

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

            'description' => 'required|string|min:10',

            'price' => 'required|numeric|min:0',

            'compare\_price' => 'nullable|numeric|gt:price',

            'is\_featured' => 'required|in:Yes,No',

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

            'category\_id' => 'required|exists:categories,id',

            'image' => 'required|image|mimes:jpeg,png,jpg,webp|max:2048',

        ], [

            'name.required' => 'The product name is required.',

            'price.numeric' => 'The price must be a number.',

            'compare\_price.gt' => 'The compare price must be greater than the actual price.',

            'image.image' => 'The uploaded file must be an image.',

            'image.max' => 'The image must not be larger than 2MB.',

            'category\_id.exists' => 'The selected category does not exist.',

            'category\_id.required' => 'The category is required.',

        ]);

        if ($validator->fails()) {

            return response()->json([

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

            ], 422);

        }

        $data = $request->all();

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

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

        }

        $product = Product::create($data);

        return new ProductResource($product);

    }

    /\*\*

     \* Display the specified resource.

     \*/

    public function show(Product $product)

    {

        return new ProductResource($product);

    }

    /\*\*

     \* Update the specified resource in storage.

     \*/

    public function update(Request $request, Product $product)

    {

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

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

            'description' => 'required|string|min:10',

            'price' => 'required|numeric|min:0',

            'compare\_price' => 'nullable|numeric|gt:price',

            'is\_featured' => 'required|in:Yes,No',

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

            'category\_id' => 'required|exists:categories,id',

            'image' => 'nullable|image|mimes:jpeg,png,jpg,webp|max:2048',

        ], [

            'name.required' => 'The product name is required.',

            'price.numeric' => 'The price must be a number.',

            'compare\_price.gt' => 'The compare price must be greater than the actual price.',

            'image.image' => 'The uploaded file must be an image.',

            'image.max' => 'The image must not be larger than 2MB.',

            'category\_id.exists' => 'The selected category does not exist.',

            'category\_id.required' => 'The category is required.',

        ]);

        if ($validator->fails()) {

            return response()->json([

                'message' => 'Validation errors',

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

            ], 422);

        }

        // Prepare data for update, including all validated fields

        $data = $request->only([

            'name',

            'description',

            'price',

            'compare\_price',

            'is\_featured',

            'status',

            'category\_id'

        ]);

        // Handle image upload

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

            // Delete old image if exists

            if ($product->image && Storage::disk('public')->exists($product->image)) {

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

            }

            // Store new image

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

        }

        // Update the product with all data

        $product->update($data);

        return (new ProductResource($product))->response()->setStatusCode(200);

    }

    /\*\*

     \* Remove the specified resource from storage.

     \*/

    public function destroy(Product $product)

    {

        try {

            // Delete image if exists

            if ($product->image && Storage::disk('public')->exists($product->image)) {

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

            }

            // Delete product

            $product->delete();

            return response()->json([

                'message' => 'Product deleted successfully.'

            ], 200);

        } catch (Exception $e) {

            return response()->json([

                'message' => 'Failed to delete the product.',

                'error' => $e->getMessage(),

            ], 500);

        }

    }

}

Inside ProductResource:

 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,

            'description' => $this->description,

            'price' => $this->price,

            'compare\_price' => $this->compare\_price ?? 'N/A',

            'is\_featured' => $this->is\_featured,

            'category' => $this->category->name,

            'category\_id' => $this->category\_id,

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

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

        ];

    }

Inside ActiveCategoryResource

public function toArray(Request $request): array

    {

        return [

            'id' => $this->id,

            'title' => $this->name,

            'value' => $this->id,

            'status' => $this->status,

        ];

    }

Inside Product Model:

// use slug instead of id

    public function getRouteKeyName()

    {

        return 'slug';

    }

Inside api.php

// products

    Route::apiResource('products', ProductController::class);

## **Inside src/stores/product.js:**

import { defineStore } from 'pinia';

import { ref } from 'vue';

import api from '@/axios';

import { showToast } from '@/utils/toast'; // Import toast utility

export const useProductStore = defineStore('product', () => {

    // ========= State =========

    const products = ref([]); // List of products

    const currentProduct = ref(null); // Single product for view/edit

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

    const loading = ref(false); // Loading state

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

    // ======= Fetch all products (GET /api/products) =======

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

        try {

            loading.value = true;

            const params = {

                page,

                per\_page: perPage,

                sort\_by: sortBy,

                sort\_direction: sortDirection,

            };

            if (search) params.search = encodeURIComponent(search);

            if (status) params.status = encodeURIComponent(status);

            const response = await api.get('/products', { params });

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

            products.value = response.data.data || [];

            pagination.value = {

                total: response.data.meta?.total || 0,

                current\_page: response.data.meta?.current\_page || 1,

                last\_page: response.data.meta?.last\_page || 1,

                per\_page: response.data.meta?.per\_page || perPage,

            };

            return response.data;

        } catch (error) {

            console.error('Fetch Products Error:', error.response?.data || error.message);

            products.value = [];

            pagination.value = { total: 0, current\_page: 1, last\_page: 1, per\_page: perPage };

            throw error;

        } finally {

            loading.value = false;

        }

    };

    // ======= Fetch a single product (GET /api/products/{slug}) =======

    const fetchProduct = async (slug) => {

        try {

            loading.value = true;

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

            currentProduct.value = response.data.data;

            return response;

        } catch (error) {

            console.error('Fetch Product Error:', error.response?.data || error.message);

            productErrors.value = error.response?.data?.errors || { general: ['Product not found'] };

            currentProduct.value = null;

            throw error;

        } finally {

            loading.value = false;

        }

    };

    // ======= Store a new product (POST /api/products) =======

    const storeProduct = async (formData) => {

        loading.value = true;

        productErrors.value = {};

        try {

            const response = await api.post('/products', formData, {

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

            });

            showToast.success('Product added successfully!');

        } catch (error) {

            const errorMsg = error.response?.data?.message || error.message;

            console.error('Store Product Error:', errorMsg);

            productErrors.value = error.response?.data?.errors || { general: [errorMsg] };

            throw error;

        } finally {

            loading.value = false;

        }

    };

    // ======= Update an existing product (PUT /api/products/{id}) =======

    const updateProduct = async (slug, formData) => {

        try {

            loading.value = true;

            productErrors.value = {};

            formData.append('\_method', 'PUT'); // Now works because formData is a FormData instance

            const response = await api.post(`/products/${slug}`, formData, {

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

            });

            showToast.success('Product updated successfully!');

        } catch (error) {

            console.error('Update Product Error:', error.response?.data || error.message);

            productErrors.value = error.response?.data?.errors || { general: ['Failed to update product'] };

            throw error;

        } finally {

            loading.value = false;

        }

    };

    // ======= Delete a product (DELETE /api/products/{id}) =======

    const deleteProduct = async (slug) => {

        try {

            loading.value = true;

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

            products.value = products.value.filter(prod => prod.slug !== slug);

            if (currentProduct.value?.slug === slug) currentProduct.value = null;

            showToast.success('Product deleted successfully!'); // Use vue3-toastify

            return;

        } catch (error) {

            console.error('Delete Product Error:', error.response?.data || error.message);

            productErrors.value = error.response?.data?.errors || { general: ['Failed to delete product'] };

            throw error;

        } finally {

            loading.value = false;

        }

    };

    // ======= Reset errors =======

    const resetErrors = () => {

        productErrors.value = {};

    };

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

    return {

        products,

        currentProduct,

        productErrors,

        loading,

        pagination,

        fetchProducts,

        fetchProduct,

        storeProduct,

        updateProduct,

        deleteProduct,

        resetErrors,

    };

});

Inside Category.js add below code:

// Fetch active categories (GET /api/categories/active)

    const fetchActiveCategories = async () => {

        try {

            loading.value = true;

            const response = await api.get('/category/active');

            activeCategories.value = response.data.data || [];

        } catch (error) {

            console.error('Fetch Active Categories Error:', error.response?.data || error.message);

            activeCategories.value = [];

            throw error;

        } finally {

            loading.value = false;

        }

    };

Inside Product.vue

<script setup>

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

import { useProductStore } from '@/stores/product'; // Adjust the path to your store

import debounce from 'lodash/debounce';

import { headers } from './utils';

// Pinia store

const productStore = useProductStore();

const itemsPerPage = ref(10);

const sortBy = ref([{ key: 'created\_at', order: 'desc' }]);

const statusFilter = ref('');

const selectedImage = ref(null); // For modal image display

const dialog = ref(false); // Modal visibility

const deleteDialog = ref(false); // Delete confirmation dialog

const productToDelete = ref(null); // Track product to delete

const search = ref(''); // Search term

// Debounced load function

const loadItems = debounce(async (options) => {

    const page = options.page || 1;

    const perPage = options.itemsPerPage || itemsPerPage.value;

    const sort = options.sortBy?.[0] || { key: 'created\_at', order: 'desc' };

    try {

        await productStore.fetchProducts(

            page,

            perPage,

            search.value,

            sort.key,

            sort.order,

            statusFilter.value

        );

    } catch (error) {

        console.error('Failed to load products:', error);

    }

}, 300);

// Initial load

onMounted(() => {

    loadItems({ page: 1, itemsPerPage: itemsPerPage.value, sortBy: sortBy.value });

});

// Watch for search changes

watch([search, statusFilter], () => {

    loadItems({ page: 1, itemsPerPage: itemsPerPage.value, sortBy: sortBy.value });

});

// Status chip color

const getStatusColor = (status) => {

    return status === 'active' ? 'green' : 'grey';

};

// Open image modal

const openImageModal = (imageUrl) => {

    selectedImage.value = imageUrl;

    dialog.value = true;

};

// Show delete confirmation

const confirmDelete = (product) => {

    productToDelete.value = product;

    deleteDialog.value = true;

};

// Delete product

const deleteProduct = async () => {

    try {

        await productStore.deleteProduct(productToDelete.value.slug);

        deleteDialog.value = false;

        productToDelete.value = null;

    } catch (error) {

        console.error('Failed to delete product:', error);

    }

};

</script>

<template>

    <v-container class="py-6">

        <!-- Filters and Search -->

        <v-card class="pa-6 mb-6 rounded-xl elevation-3">

            <v-row justify="space-between" no-gutters>

                <v-col cols="12" sm="6" md="4" class="pa-2">

                    <v-text-field v-model="search" label="Search Products" prepend-inner-icon="mdi-magnify"

                        variant="outlined" clearable density="comfortable" class="rounded-lg" bg-color="white" flat />

                </v-col>

                <v-col cols="12" sm="6" md="4" class="pa-2">

                    <v-select v-model="statusFilter" :items="[

                        { title: 'All', value: '' },

                        { title: 'Active', value: 'active' },

                        { title: 'Inactive', value: 'inactive' }

                    ]" label="Filter by Status" prepend-inner-icon="mdi-filter" variant="outlined"

                        density="comfortable" class="rounded-lg" bg-color="white" flat clearable />

                </v-col>

                <v-col cols="12" sm="6" md="4" class="pa-2 text-right mt-2">

                    <router-link :to="{ name: 'AddProduct' }" class="text-decoration-none">

                        <v-btn color="teal-darken-2" class="rounded-lg px-6" elevation="2">

                            <v-icon left class="mr-2">mdi-plus</v-icon>

                            Add Product

                        </v-btn>

                    </router-link>

                </v-col>

            </v-row>

        </v-card>

        <!-- Data Table Server -->

        <v-card class="rounded-lg elevation-2">

            <v-data-table-server v-model:items-per-page="itemsPerPage" :headers="headers" :items="productStore.products"

                :items-length="productStore.pagination.total || 0" :loading="productStore.loading" item-value="id"

                v-model:sort-by="sortBy" :items-per-page-options="[5, 10, 20, 50]" @update:options="loadItems"

                height="550" fixed-header>

                <!-- S:N column -->

                <template v-slot:item.sn="{ index }">

                    {{ (productStore.pagination.current\_page - 1) \* itemsPerPage + index + 1 }}

                </template>

                <!-- Image column -->

                <template v-slot:item.image="{ item }">

                    <v-img v-if="item.image" :src="item.image" height="40" width="40"

                        class="my-2 rounded-lg cursor-pointer" @click="openImageModal(item.image)" />

                    <span v-else>No Image</span>

                </template>

                <!-- Status column -->

                <template v-slot:item.status="{ item }">

                    <v-chip :color="getStatusColor(item.status)" small class="text-uppercase font-weight-bold">

                        {{ item.status }}

                    </v-chip>

                </template>

                <!-- Actions column -->

                <template v-slot:item.actions="{ item }">

                    <router-link :to="{ name: 'EditProduct', params: { slug: item.slug } }">

                        <v-btn icon color="warning" size="small">

                            <v-icon>mdi-pencil</v-icon>

                        </v-btn>

                    </router-link>

                    <v-btn icon color="red-darken-2" size="small" class="ms-2" @click="confirmDelete(item)">

                        <v-icon>mdi-delete</v-icon>

                    </v-btn>

                </template>

            </v-data-table-server>

        </v-card>

        <!-- Image Modal -->

        <v-dialog v-model="dialog" max-width="500">

            <v-card class="rounded-lg elevation-3">

                <v-card-title class="d-flex align-center" style="background: linear-gradient(45deg, #00695c, #00897b)">

                    <span class="text-h6 font-weight-bold text-white">Product Image</span>

                    <v-spacer />

                    <v-btn icon @click="dialog = false" variant="plain">

                        <v-icon color="white">mdi-close</v-icon>

                    </v-btn>

                </v-card-title>

                <v-card-text class="text-center py-4">

                    <v-img :src="selectedImage" max-height="400" contain class="mx-auto rounded-lg" />

                </v-card-text>

            </v-card>

        </v-dialog>

        <!-- Delete Confirmation Dialog -->

        <v-dialog v-model="deleteDialog" max-width="400px">

            <v-card class="rounded-lg elevation-4">

                <v-card-title class="text-h6 font-weight-bold pa-4"

                    style="background: linear-gradient(45deg, #d32f2f, #f44336); color: white;">

                    <v-icon color="white" class="mr-2">mdi-alert-circle</v-icon>

                    Confirm Deletion

                </v-card-title>

                <v-card-text class="pa-6 text-body-1 text-grey-darken-2">

                    Are you sure you want to delete the product "<strong>{{ productToDelete?.name }}</strong>"? This

                    action

                    cannot be undone.

                </v-card-text>

                <v-card-actions class="pa-4">

                    <v-spacer />

                    <v-btn color="grey-darken-1" variant="outlined" class="rounded-md" @click="deleteDialog = false">

                        Cancel

                    </v-btn>

                    <v-btn color="red-darken-2" variant="flat" class="rounded-md ml-2" @click="deleteProduct">

                        <v-icon left>mdi-delete</v-icon>

                        Delete

                    </v-btn>

                </v-card-actions>

            </v-card>

        </v-dialog>

    </v-container>

</template>

<style scoped>

.cursor-pointer {

    cursor: pointer;

}

</style>

Inside AddProduct.vue

<script setup>

import { ref, onMounted } from 'vue';

import { useProductStore } from '@/stores/product'; // Adjust the path to your store

import { useRouter } from 'vue-router';

import { useCategoryStore } from '@/stores/category'; // Adjust path

// Pinia store and router

const productStore = useProductStore();

const categoryStore = useCategoryStore();

const router = useRouter();

// Form data

const form = ref({

    name: '',

    description: '',

    price: '',

    compare\_price: '',

    is\_featured: 'No',

    status: 'active',

    category\_id: '',

    image: null,

});

// Validation rules

const rules = {

    name: [

        v => !!v || 'Product name is required',

        v => (v && v.length <= 255) || 'Name must be less than 255 characters',

    ],

    description: [

        v => !!v || 'Description is required',

        v => (v && v.length >= 10) || 'Description must be at least 10 characters',

    ],

    price: [

        v => !!v || 'Price is required',

        v => (!isNaN(v) && v >= 0) || 'Price must be a positive number',

    ],

    compare\_price: [

        v => (!v || (!isNaN(v) && Number(v) > Number(form.value.price)) || 'Compare price must be greater than price')],

    is\_featured: [v => !!v || 'Featured status is required'],

    status: [v => !!v || 'Status is required'],

    category\_id: [v => !!v || 'Category is required'],

    image: [v => !!v || 'Image is required'],

};

// Form reference for validation

const formRef = ref(null);

// Error messages from backend

const backendErrors = ref({});

// Fetch active categories on mount

onMounted(async () => {

    try {

        await categoryStore.fetchActiveCategories();

    } catch (error) {

        console.error('Failed to fetch active categories:', error);

    }

});

// Handle file input

const onFileChange = (event) => {

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

};

// Submit form

const submitForm = async () => {

    const { valid } = await formRef.value.validate();

    if (!valid) return;

    try {

        backendErrors.value = {}; // Clear previous errors

        await productStore.storeProduct(form.value);

        router.push({ name: 'Product' }); // Redirect to product list page

    } catch (error) {

        backendErrors.value = error.response?.data?.errors || { general: ['Failed to add product'] };

    }

};

// Reset form

const resetForm = () => {

    form.value = {

        name: '',

        description: '',

        price: '',

        compare\_price: '',

        is\_featured: 'No',

        status: 'active',

        category\_id: '',

        image: null,

    };

    backendErrors.value = {};

    formRef.value.resetValidation();

};

</script>

<template>

    <v-container fluid class="mt-3">

        <v-row justify="center">

            <v-col cols="12">

                <v-card class="pa-6 rounded-xl elevation-4">

                    <v-card-title class="text-h5 font-weight-bold teal-darken-2">

                        <v-icon left size="large" class="mr-2">mdi-package-variant-closed</v-icon>

                        Add New Product

                    </v-card-title>

                    <v-card-text>

                        <!-- General error from backend -->

                        <v-alert v-if="backendErrors.general" type="error" variant="tonal" class="mb-4 rounded-lg">

                            {{ backendErrors.general[0] }}

                        </v-alert>

                        <v-form ref="formRef" @submit.prevent="submitForm">

                            <!-- Product Name -->

                            <v-text-field v-model="form.name" label="Product Name" prepend-inner-icon="mdi-label"

                                variant="outlined" :rules="rules.name" :error-messages="backendErrors.name"

                                class="rounded-lg mb-4" bg-color="white" />

                            <!-- Product Description -->

                            <v-textarea v-model="form.description" label="Description" prepend-inner-icon="mdi-text"

                                variant="outlined" :rules="rules.description"

                                :error-messages="backendErrors.description" class="rounded-lg mb-4" bg-color="white" />

                            <v-row>

                                <v-col cols="12" sm="6">

                                    <!-- Product Price -->

                                    <v-text-field v-model="form.price" label="Price"

                                        prepend-inner-icon="mdi-currency-usd" variant="outlined" type="number"

                                        :rules="rules.price" :error-messages="backendErrors.price"

                                        class="rounded-lg mb-4" bg-color="white" />

                                </v-col>

                                <v-col cols="12" sm="6">

                                    <!-- Product Compared Price -->

                                    <v-text-field v-model="form.compare\_price" label="Compare Price"

                                        prepend-inner-icon="mdi-currency-usd" variant="outlined" type="number"

                                        :rules="rules.compare\_price" :error-messages="backendErrors.compare\_price"

                                        class="rounded-lg mb-4" bg-color="white" />

                                </v-col>

                            </v-row>

                            <!-- Product Is Featured Yes/No -->

                            <v-select v-model="form.is\_featured" :items="['Yes', 'No']" label="Is Featured"

                                prepend-inner-icon="mdi-star" variant="outlined" :rules="rules.is\_featured"

                                :error-messages="backendErrors.is\_featured" class="rounded-lg mb-4" bg-color="white" />

                            <!-- Product Status -->

                            <v-select v-model="form.status" :items="['active', 'inactive']" label="Status"

                                prepend-inner-icon="mdi-toggle-switch" variant="outlined" :rules="rules.status"

                                :error-messages="backendErrors.status" class="rounded-lg mb-4" bg-color="white" />

                            <!-- Product Select Category -->

                            <v-select label="Category" v-model="form.category\_id"

                                :items="categoryStore.activeCategories" prepend-inner-icon="mdi-shape"

                                variant="outlined" :rules="rules.category\_id"

                                :error-messages="backendErrors.category\_id" class="rounded-lg mb-4"

                                :loading="categoryStore.loading" />

                            <!-- Product Image -->

                            <v-file-input v-model="form.image" label="Product Image" prepend-inner-icon="mdi-camera"

                                variant="outlined" accept="image/jpeg,image/png,image/jpg,image/webp"

                                :rules="rules.image" :error-messages="backendErrors.image" class="rounded-lg mb-4"

                                bg-color="white" @change="onFileChange" />

                            <v-row justify="end">

                                <!-- Reset Form -->

                                <v-btn color="grey-darken-1" variant="outlined" class="rounded-lg mr-4"

                                    @click="resetForm">

                                    <v-icon left class="mr-2">mdi-undo</v-icon>

                                    Reset

                                </v-btn>

                                <!-- Submit Form -->

                                <v-btn color="teal-darken-2" variant="flat" class="rounded-lg" type="submit"

                                    :loading="productStore.loading">

                                    <v-icon left class="mr-2">mdi-content-save</v-icon>

                                    Save Product

                                </v-btn>

                            </v-row>

                        </v-form>

                    </v-card-text>

                </v-card>

            </v-col>

        </v-row>

    </v-container>

</template>

Inside EditProduct.vue

<script setup>

import { ref, onMounted } from 'vue';

import { useProductStore } from '@/stores/product'; // Adjust path to your store

import { useCategoryStore } from '@/stores/category'; // Adjust path

import { useRouter, useRoute } from 'vue-router';

// Pinia stores and router

const productStore = useProductStore();

const categoryStore = useCategoryStore();

const router = useRouter();

const route = useRoute();

// Form data

const form = ref({

    name: '',

    description: '',

    price: '',

    compare\_price: '',

    is\_featured: 'No',

    status: 'active',

    category\_id: '',

    image: null,

});

// Validation rules (same as AddProduct.vue)

const rules = {

    name: [

        v => !!v || 'Product name is required',

        v => (v && v.length <= 255) || 'Name must be less than 255 characters',

    ],

    description: [

        v => !!v || 'Description is required',

        v => (v && v.length >= 10) || 'Description must be at least 10 characters',

    ],

    price: [

        v => !!v || 'Price is required',

        v => (!isNaN(v) && v >= 0) || 'Price must be a positive number',

    ],

    compare\_price: [

        v => (!v || (!isNaN(v) && Number(v) > Number(form.value.price)) || 'Compare price must be greater than price')],

    is\_featured: [v => !!v || 'Featured status is required'],

    status: [v => !!v || 'Status is required'],

    category\_id: [v => !!v || 'Category is required'],

    image: [v => !form.value.image || !!v || 'Image is required'], // Optional for updates

};

// Form reference for validation

const formRef = ref(null);

// Error messages from backend

const backendErrors = ref({});

// Fetch product and categories on mount

onMounted(async () => {

    try {

        // Fetch active categories

        await categoryStore.fetchActiveCategories();

        // Fetch product details using slug from route

        const slug = route.params.slug;

        await productStore.fetchProduct(slug);

        // Populate form with existing product data

        const product = productStore.currentProduct;

        if (product) {

            form.value = {

                name: product.name,

                description: product.description,

                price: product.price,

                compare\_price: product.compare\_price || '',

                is\_featured: product.is\_featured,

                status: product.status,

                category\_id: product.category\_id,

                image: null, // Image is not pre-filled; user can upload a new one

            };

        }

    } catch (error) {

        console.error('Failed to load data:', error);

    }

});

// Handle file input

const onFileChange = (event) => {

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

};

// Submit form to update product

const submitForm = async () => {

    const { valid } = await formRef.value.validate();

    if (!valid) return;

    try {

        backendErrors.value = {}; // Clear previous errors

        const slug = route.params.slug;

        // Create FormData instance

        const formData = new FormData();

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

        formData.append('description', form.value.description);

        formData.append('price', form.value.price);

        if (form.value.compare\_price) formData.append('compare\_price', form.value.compare\_price);

        formData.append('is\_featured', form.value.is\_featured);

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

        formData.append('category\_id', form.value.category\_id);

        if (form.value.image) formData.append('image', form.value.image); // Append image only if provided

        await productStore.updateProduct(slug, formData);

        router.push({ name: 'Product' }); // Redirect to product list page

    } catch (error) {

        backendErrors.value = error.response?.data?.errors || { general: ['Failed to update product'] };

    }

};

// Reset form to original product data

const resetForm = () => {

    const product = productStore.currentProduct;

    if (product) {

        form.value = {

            name: product.name,

            description: product.description,

            price: product.price,

            compare\_price: product.compare\_price || '',

            is\_featured: product.is\_featured,

            status: product.status,

            category\_id: product.category\_id,

            image: null,

        };

    }

    backendErrors.value = {};

    formRef.value.resetValidation();

};

</script>

<template>

    <v-container fluid class="mt-3">

        <v-row justify="center">

            <v-col cols="12">

                <v-card class="pa-6 rounded-xl elevation-4">

                    <v-card-title class="text-h5 font-weight-bold teal-darken-2">

                        <v-icon left size="large" class="mr-2">mdi-package-variant-closed</v-icon>

                        Edit Product

                    </v-card-title>

                    <v-card-text>

                        <!-- General error from backend -->

                        <v-alert v-if="backendErrors.general" type="error" variant="tonal" class="mb-4 rounded-lg">

                            {{ backendErrors.general[0] }}

                        </v-alert>

                        <v-form ref="formRef" @submit.prevent="submitForm">

                            <!-- Product Name -->

                            <v-text-field v-model="form.name" label="Product Name" prepend-inner-icon="mdi-label"

                                variant="outlined" :rules="rules.name" :error-messages="backendErrors.name"

                                class="rounded-lg mb-4" bg-color="white" />

                            <!-- Product Description -->

                            <v-textarea v-model="form.description" label="Description" prepend-inner-icon="mdi-text"

                                variant="outlined" :rules="rules.description"

                                :error-messages="backendErrors.description" class="rounded-lg mb-4" bg-color="white" />

                            <v-row>

                                <v-col cols="12" sm="6">

                                    <!-- Product Price -->

                                    <v-text-field v-model="form.price" label="Price"

                                        prepend-inner-icon="mdi-currency-usd" variant="outlined" type="number"

                                        :rules="rules.price" :error-messages="backendErrors.price"

                                        class="rounded-lg mb-4" bg-color="white" />

                                </v-col>

                                <v-col cols="12" sm="6">

                                    <!-- Product Compared Price -->

                                    <v-text-field v-model="form.compare\_price" label="Compare Price"

                                        prepend-inner-icon="mdi-currency-usd" variant="outlined" type="number"

                                        :rules="rules.compare\_price" :error-messages="backendErrors.compare\_price"

                                        class="rounded-lg mb-4" bg-color="white" />

                                </v-col>

                            </v-row>

                            <!-- Product Is Featured Yes/No -->

                            <v-select v-model="form.is\_featured" :items="['Yes', 'No']" label="Is Featured"

                                prepend-inner-icon="mdi-star" variant="outlined" :rules="rules.is\_featured"

                                :error-messages="backendErrors.is\_featured" class="rounded-lg mb-4" bg-color="white" />

                            <!-- Product Status -->

                            <v-select v-model="form.status" :items="['active', 'inactive']" label="Status"

                                prepend-inner-icon="mdi-toggle-switch" variant="outlined" :rules="rules.status"

                                :error-messages="backendErrors.status" class="rounded-lg mb-4" bg-color="white" />

                            <!-- Product Select Category -->

                            <v-select label="Category" v-model="form.category\_id"

                                :items="categoryStore.activeCategories" prepend-inner-icon="mdi-shape"

                                variant="outlined" :rules="rules.category\_id"

                                :error-messages="backendErrors.category\_id" class="rounded-lg mb-4"

                                :loading="categoryStore.loading" />

                            <!-- Product Image -->

                            <v-file-input v-model="form.image" label="Product Image" prepend-inner-icon="mdi-camera"

                                variant="outlined" accept="image/jpeg,image/png,image/jpg,image/webp"

                                :rules="rules.image" :error-messages="backendErrors.image" class="rounded-lg mb-4"

                                bg-color="white" @change="onFileChange" />

                            <v-row justify="end">

                                <!-- Reset Form -->

                                <v-btn color="grey-darken-1" variant="outlined" class="rounded-lg mr-4"

                                    @click="resetForm">

                                    <v-icon left class="mr-2">mdi-undo</v-icon>

                                    Reset

                                </v-btn>

                                <!-- Submit Form -->

                                <v-btn color="teal-darken-2" variant="flat" class="rounded-lg" type="submit"

                                    :loading="productStore.loading">

                                    <v-icon left class="mr-2">mdi-content-save</v-icon>

                                    Update Product

                                </v-btn>

                            </v-row>

                        </v-form>

                    </v-card-text>

                </v-card>

            </v-col>

        </v-row>

    </v-container>

</template>

Inside index.js

 {

                path: 'product',

                name: 'Product',

                component: () => import('@/views/backend/products/Product.vue'),

            },

            {

                path: 'product/add',

                name: 'AddProduct',

                component: () => import('@/views/backend/products/AddProduct.vue'),

            },

            {

                path: 'product/edit/:slug',

                name: 'EditProduct',

                component: () => import('@/views/backend/products/EditProduct.vue'),

            },