1) Authentication Using Laravel Sanctum (Token Based Authentication)

1) Install necessary packages:

composer require laravel/sanctum (if not already installed)
php artisan vendor:publish --provider="Laravel\Sanctum\SanctumServiceProvider"
php artisan migrate

2) Inside api.php

```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']);
});
```

3) Create API/AuthController:

```php
<?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;
```

```php
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());
    }
}
```

4) 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,
            ]);
    }
}
```

5) Inside .env

```
MAIL_MAILER=smtp
MAIL_HOST=smtp.gmail.com
MAIL_PORT=465
MAIL_USERNAME=freelenceruniттesting@gmail.com
MAIL_PASSWORD=aqcdacybfhqsyvsc
MAIL_ENCRYPTION=tls
MAIL_FROM_ADDRESS="freelenceruniттesting@gmail.com"
MAIL_FROM_NAME="${APP_NAME}"
```

6) 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>
```

7) Inside .env

```
FRONTEND_URL=http://localhost:3000
SESSION_DOMAIN=localhost
SANCTUM_STATEFUL_DOMAINS=localhost:3000
```

Setup VueJs with Vuetify:

1) Install below necessary packages:

```
npm create vite@latest
npm install
npm install vue-router@4
npm install pinia
npm install axios
```

Clear App.Vue and delete HelloWorld.vue and delete files inside assets.
Create src/router/index.js

2) 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
    }
  }
})
```

3) Setup Vuetify:

```
npm install vuetify@latest
```

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:

```
npm install @mdi/font
```

```
// src/main.js
import '@mdi/font/css/materialdesignicons.css';
```

```
<v-btn icon>
  <v-icon>mdi-home</v-icon>
</v-btn>
```

Now, you can use icon and vuetify.

4) 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;
```

5) 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');
```

6) 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>
```

7) 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:

1) 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
```

2) 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,
    }
})
```

3) 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
```

4) 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>
```

5) 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):

1) Inside user model

```php
use Illuminate\Contracts\Auth\MustVerifyEmail;

class User extends Authenticatable  implements MustVerifyEmail
{
```

2) Inside AuthController:

```php
<?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());
    }
}
```

3) 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}",
            ]);
    }
}
```

4) 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>
```

5) Inside users table:

```
 $table->string('verification_token')->nullable();
```

6) Inside api.php

```
<?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']);
});
```

7) Inside auth.js

```javascript
// 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,
    }
})
```

8) 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: '/admin',
        name: 'Admin',
        component: () => import('@/layouts/BackendLayout.vue'),
        meta: { requiresAuth: true },
        children: [
            {
                path: 'home',
                name: 'Dashboard',
                component: () => import('@/views/backend/Home.vue'),
            },
        ],
    },
]

const router = createRouter({
    history: createWebHistory(),
    routes,
})

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

    if (to.meta.requiresAuth && !auth.isAuthenticated) {
        return next('/login')
    }

    if (to.path === '/login' && auth.isAuthenticated) {
        return next('/admin/home')
    }

    return next()
})
```

```
export default router
```

9) 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>
```

10) 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>
```

11) 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:

1. 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>
```

2. 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.' }
        }
    }
```

3. 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:

1. 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' }
        }
    }
```

2. 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>
```

3. 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>
```

4. 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:

1. Make model and migration for category.

```php
Schema::create('categories', function (Blueprint $table) {
    $table->id();
    $table->string('slug')->unique();
    $table->string('name')->unique();
    $table->string('image');
    $table->enum('status',['active','inactive'])->default('active');
    $table->timestamps();
});
```

```php
<?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;
    }
}
```

2. Make controller api/CategoryController –r

```
<?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);
    }
```

```php
    /**
     * 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);
    }
}
```

3. 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(),
        ];
    }
```

4. 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');
    });
```