**Task 1: Request Validation**

Implement request validation for a registration form that contains the following fields: name, email, and password. Validate the following rules:

name: required, string, minimum length 2.

email: required, valid email format.

password: required, string, minimum length 8.

**ANS 01:**

To implement request validation for a registration form with fields like name, email, and password, you can follow these steps:

1. Create a new form request:

**php artisan make:request RegistrationRequest**

1. In the rules() method of the RegistrationRequest class, define the validation rules for the registration form fields:

**public function rules()**

**{**

**return [**

**'name' => 'required|string|min:2',**

**'email' => 'required|email',**

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

**];**

**}**

1. Message example:

**public function messages()**

**{**

**return [**

**'name.required' => 'The name field is required.',**

**'name.string' => 'The name must be a string.',**

**'name.min' => 'The name must have a minimum length of 2 characters.',**

**'email.required' => 'The email field is required.',**

**'email.email' => 'Please enter a valid email address.',**

**'password.required' => 'The password field is required.',**

**'password.string' => 'The password must be a string.',**

**'password.min' => 'The password must have a minimum length of 8 characters.',**

**];**

**}**

1. RegistrationRequest class in the store() method to automatically apply the request validation. For example:

**use App\Http\Requests\RegistrationRequest;**

**public function store(RegistrationRequest $request)**

**{**

**// Handle the registration logic as you want**

**}**

**Task 2: Request Redirect**

Create a route /home that redirects to /dashboard using a 302 redirect.

**ANS 02:**

**Add a route:**

**Route::get('/home', function () {**

**return Redirect::to('/dashboard', 302);**

**});**

**Task 3: Global Middleware**

Create a global middleware that logs the request method and URL for every incoming request. Log the information to the Laravel log file.

**ANS 03:**

1. Create a new middleware:

php artisan make:middleware LogRequests

1. Open the LogRequests.php file and replace the content with the following code:

**<?PHP**

**namespace App\Http\Middleware;**

**use Closure;**

**use Illuminate\Support\Facades\Log;**

**class LogRequests**

**{**

**public function handle($request, Closure $next)**

**{**

**$method = $request->method();**

**$url = $request->fullUrl();**

**Log::info("Request: [{$method}] {$url}");**

**return $next($request);**

**}**

**}**

1. Register the middleware as a global middleware in the App\Http\Kernel class. Open the app/Http/Kernel.php file.

Locate the $middleware property and add the fully qualified class name of the LogRequests middleware to the array:

**protected $middleware = [**

**// Other middleware entries...**

**\App\Http\Middleware\LogRequests::class,**

**];**

**Task 4: Route Middleware**

Create a route group for authenticated users only. This group should include routes for /profile and /settings. Apply a middleware called AuthMiddleware to the route group to ensure only authenticated users can access these routes.

**ANS 04**:

1. Create route as following convension:

**Route::middleware([AuthMiddleware::class])->group(function () {**

**Route::get('/profile', [ProfileController::class, 'index'])->name('profile');**

**Route::get('/settings', [SettingsController::class, 'index'])->name('settings');**

**});**

**Task 5: Controller**

Create a controller called ProductController that handles CRUD operations for a resource called Product. Implement the following methods:

index(): Display a list of all products.

create(): Display the form to create a new product.

store(): Store a newly created product.

edit($id): Display the form to edit an existing product.

update($id): Update the specified product.

destroy($id): Delete the specified product.

**ANS 05:**

1. create a ProductController:

**php artisan make:controller ProductController –resource**

1. Product Controller page:

**class ProductController extends Controller**

**{**

**public function index()**

**{**

**$products = Product::all();**

**return view('products.index', compact('products'));**

**}**

**public function create()**

**{**

**return view('products.create');**

**}**

**public function store(Request $request)**

**{**

**$validatedData = $request->validate([**

**'name' => 'required',**

**'price' => 'required|numeric',**

**]);**

**$product = Product::create($validatedData);**

**return redirect()->route('products.index')->with('success', 'Product created successfully');**

**}**

**public function edit($id)**

**{**

**$product = Product::findOrFail($id);**

**return view('products.edit', compact('product'));**

**}**

**public function update(Request $request, $id)**

**{**

**$product = Product::findOrFail($id);**

**$validatedData = $request->validate([**

**'name' => 'required',**

**'price' => 'required|numeric',**

**]);**

**$product->update($validatedData);**

**return redirect()->route('products.index')->with('success', 'Product updated successfully');**

**}**

**public function destroy($id)**

**{**

**$product = Product::findOrFail($id);**

**$product->delete();**

**return redirect()->route('products.index')->with('success', 'Product deleted successfully');**

**}**

**}**

**Task 6: Single Action Controller**

Create a single action controller called ContactController that handles a contact form submission. Implement the \_\_invoke() method to process the form submission and send an email to a predefined address with the submitted data.

**ANS 06:**

1. Create controller:

**php artisan make:controller ContactController –invokable**

1. Controller file:

**class ContactController extends Controller**

**{**

**public function \_\_invoke(Request $request)**

**{**

**$validatedData = $request->validate([**

**'name' => 'required',**

**'email' => 'required|email',**

**'message' => 'required',**

**]);**

**$name = $validatedData['name'];**

**$email = $validatedData['email'];**

**$message = $validatedData['message'];**

**// in your Laravel application**

**Mail::to('your-email@example.com')->send(new ContactFormMail($name, $email, $message));**

**return redirect()->back()->with('success', 'Thank you for your message!');**

**}**

**}**

**Task 7: Resource Controller**

Create a resource controller called PostController that handles CRUD operations for a resource called Post. Ensure that the controller provides the necessary methods for the resourceful routing conventions in Laravel.

**ANS:**

1. Create controller:

**php artisan make:controller PostController –resource**

1. Controller function:

**public function index()**

**{**

**$posts = Post::all();**

**return view('posts.index', ['posts' => $posts]);**

**}**

1. Route:

**Route::resource('posts', PostController::class);**

**Task 8: Blade Template Engine**

Create a Blade view called welcome.blade.php that includes a navigation bar and a section displaying the text "Welcome to Laravel!".

**ANS 08:**

Step 01: Create a blade file name convention

**welcome.blade.php**

Step 02: Declare route:

**Route::get('/', function () {**

**return view('welcome');**

**});**