Task 1: Request Validation

At first we need create a new file DataValidation.php in the app/Rules directory by writing the following artisan command in the terminal which creates a new file in app/Rules/ directory:

php artisan make:rule DataValidation

Then we need to open the DataValidation.php file and write the following code:

```
<?php

namespace App\Rules;

use Illuminate\Contracts\Validation\Rule;

class DataValidation implements Rule

{

   public function passes($attribute, $value)

   {

      return filter_var($value, FILTER_VALIDATE_EMAIL) !== false;

   }

   public function message()

   {

      return 'The :attribute must be a valid email address.';

   }

}
```

After that we need to create a new controller to the directory app/Http/Controllers/ and I mention my new controller name as RegisterController.php by the following artisan command in the terminal:

```
php artisan make:controller RegisterController
```

Then we need to write the following code in RegisterController.php file:

```
<?php

public function register(Request $request)

{

   $validatedData = $request->validate([

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

      'email' => ['required', new DataValidation],

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

   ]);

   return response()->json(['message' => 'Registration successful'], 200);

}
```

Then we have to define the route for the registration endpoint in your routes/api.php file by following code:

```
use App\Http\Controllers\RegisterController;

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

Task 2: Request Redirect

We have to write the following code in the directory routes/web.php:

```
Route::redirect('/home', '/dashboard', 302);
```

Task 3: Creating Global Middleware

At first we need to create a middleware in the directory app/Http/Middleware and I am creating a middleware by the name LogRequestsMiddleware.php by writing the following code in the terminal:

```
php artisan make:middleware LogRequestsMiddleware
```

Then in LogRequestsMiddleware.php write the following code:

```
<?php

namespace App\Http\Middleware;

use Closure;

use Illuminate\Support\Facades\Log;

class LogRequestsMiddleware

{

   public function handle($request, Closure $next)

   {

      Log::info('Request: ' . $request->method() . ' ' . $request->fullUrl());

      return $next($request);

   }

}
```

Then we need to append the following code inside the middleware property in the app/Http/Kernel.php file:

```
protected $middleware = [

   \App\Http\Middleware\LogRequestsMiddleware::class,

];
```

Task 4: Route Middleware

At first, let's create the AuthMiddleware using the Artisan command line tool:

php artisan make:middleware AuthMiddleware

This will generate a new middleware file called AuthMiddleware in the app/Http/Middleware directory. We need to open the file and update the handle method as follows:

```
<?php

namespace App\Http\Middleware;

use Closure;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Auth;


class AuthMiddleware

{

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

   {

      if (Auth::check()) {

         return $next($request);

      }

      return redirect('/login'); // Redirect to the login page if user is not authenticated

   }

}
```

Now we have to open the routes/web.php file and define the route group for authenticated users:

```
<?php

use Illuminate\Support\Facades\Route;
```

```
Route::middleware(['auth'])->group(function () {
    Route::get('/profile', 'ProfileController@index')->name('profile');
    Route::get('/settings', 'SettingsController@index')->name('settings');
});
```

In this example, we assume that the ProfileController and SettingsController have been created and contain the respective methods index for displaying the profile and settings pages.

Now need to open the app/Http/Kernel.php file and add the AuthMiddleware to the $routeMiddleware property:

```
protected $routeMiddleware = [
    // Other middleware...
    'auth' => \App\Http\Middleware\AuthMiddleware::class,
];
```

To protect the routes /profile and /settings, we need to make authentication scaffolding using the make:auth by following Artisan command:

```
php artisan make:auth
```

This command will generate the necessary authentication views, routes, and controllers.

Task 5: Controller task for CRUD

To create the ProductController with CRUD operations for the Product resource, follow these steps:

Open your Laravel project and navigate to the app/Http/Controllers directory.

Inside the Controllers directory, we need to create a new file called ProductController.php. Then we need to write the following code in the file:

```php
<?php
namespace App\Http\Controllers;
use App\Models\Product;
use Illuminate\Http\Request;
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',
            'description' => '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)
    {
        $validatedData = $request->validate([
            'name' => 'required',
            'description' => 'required',
            'price' => 'required|numeric',
        ]);
        $product = Product::findOrFail($id);
        $product->update($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');
    }
}
```

Now we need to define the routes to map to these controller methods in the routes/web.php file as follow:

```
use App\Http\Controllers\ProductController;

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

Route::get('/products/create', [ProductController::class, 'create'])->name('products.create');

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

Route::get('/products/{id}/edit', [ProductController::class, 'edit'])->name('products.edit');

Route::put('/products/{id}', [ProductController::class, 'update'])->name('products.update');

Route::delete('/products/{id}', [ProductController::class, 'destroy'])->name('products.destroy');
```

Make sure to adjust the routes as per your desired URL structure and naming conventions.

Task 6: Single Action Controller

Inside the Controllers directory app/Http/Controllers directory, we need to create a new file called ContactController.php.

Now we need to write the following code in the ContactController.php file:

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Mail;

class ContactController extends Controller

{

    public function __invoke(Request $request)

    {

        $validatedData = $request->validate([

            'name' => 'required',
```

```
            'email' => 'required|email',
            'message' => 'required',
        ]);
        Mail::to('your-email@example.com')->send(new ContactFormMail($validatedData));
        return response()->json(['message' => 'Form submitted successfully']);
    }
}
```

Create a new Mailable class for sending the contact form email by writing following terminal command:

```
php artisan make:mail ContactFormMail
```

Now open the app/Mail/ContactFormMail.php file and need to modify it to match our contact form email content as follows:

```
<?php
namespace App\Mail;
use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Mail\Mailable;
use Illuminate\Queue\SerializesModels;
class ContactFormMail extends Mailable
{
    use Queueable, SerializesModels;
    public $formData;
    public function __construct($formData)
    {
        $this->formData = $formData;
    }
```

```
    public function build()
    {
        return $this->view('emails.contact-form')
                    ->with(['formData' => $this->formData])
                    ->subject('Contact Form Submission');
    }
}
```

Here we pass the submitted form data to the Mailable class through the constructor. The build() method sets the email view, data, and subject.

Task 7: Resource Controller

Inside the Controllers directory app/Http/Controllers directory, we need to create a new file called PostController.php.

Now we need to write the following code in the PostController.php file:

```
<?php
namespace App\Http\Controllers;
use App\Models\Post;
use Illuminate\Http\Request;
class PostController extends Controller
{
    public function index()
    {
        $posts = Post::all();
        return view('posts.index', compact('posts'));
    }
```

```
public function create()
{
    return view('posts.create');
}
public function store(Request $request)
{
    $validatedData = $request->validate([
        'title' => 'required|string',
        'content' => 'required|string',
    ]);
    $post = Post::create($validatedData);
    return redirect()->route('posts.show', ['post' => $post->id]);
}
public function show(Post $post)
{
    return view('posts.show', compact('post'));
}
public function edit(Post $post)
{
    return view('posts.edit', compact('post'));
}
public function update(Request $request, Post $post)
{
    $validatedData = $request->validate([
        'title' => 'required|string',
```

```
            'content' => 'required|string',
        ]);
        $post->update($validatedData);
        return redirect()->route('posts.show', ['post' => $post->id]);
    }
    public function destroy(Post $post)
    {
        $post->delete();
        return redirect()->route('posts.index')->with('success', 'Post deleted successfully');
    }
}
```

Now we need to define the corresponding routes in the routes/web.php file by writing following code:

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

Task 8: Blade Template Engine

We need to create a new file called welcome.blade.php in to the resources/views directory by following artisan command:

```
touch resources/views/welcome.blade.php
```

Now we need to open the welcome.blade.php file and write the following code:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
```

```
  <title>Portfolio</title>
  </head><body>
    <!-- Navigation bar -->
    <nav>
        <!-- Your navigation bar HTML goes here -->
    </nav>

    <!-- Content section -->
    <section>
        <h1>Welcome to Laravel!</h1>
        <!-- Additional content goes here -->
    </section>
</body>
</html>
```