PHP-INTRODUCTION OF LARAVEL PHP FRAMEWORK-INDUSTRY

1) Create Following Routes.(for frontend side application)

a) Home page

b) About us

c) Contact us

d) Gallery

e) Registration

f) Login

Ans:-

a) Home page:-

```
Route::get('/index', function () {
   return view('website/index');
});
```

b) About us:-

```
Route::get('/about', function () {
   return view('website/about');
});
```

c) Contact us:-

```
Route::get('/contact', function () {
   return view('website/contact');
});
```

d) Gallery:-

```
Route::get('/gallery', function () {
   return view('website/gallery');
});
```

e) Registration:-

```
Route::get('/signup', function () {
   return view('website/signup');
});
```

f) Login:-

```
 Route::get('/login', function () {
   return view('website/login');
});
```

2) How to pass Multiple Variable in route?

Ans:-

- ***Define the Route with Multiple Parameters:***

In your '`routes/web.php`' file, define a route with multiple parameters. You can use curly braces '`{}`' to specify the parameters.

```
Route::get('/example/{param1}/{param2}', 'ExampleController@index');
```

In this example, we have defined a route with two parameters: '`param1`' and '`param2`'.

- ***Create a Controller:***

You should create a controller (if you haven't already) to handle this route. You can generate a controller using the '`artisan`' command-line tool:

php artisan make:controller ExampleController

This will create an '`ExampleController.php`' file in the '`app/Http/Controllers`' directory.

- ## *Handle the Route Parameters in the Controller:*

In the '`ExampleController`', define the '`index`' method and capture the route parameters using the same names you defined in the route. For example:

```
namespace App\Http\Controllers;

use Illuminate\Http\Request;

class ExampleController extends Controller
{
   public function index($param1, $param2)
   {
      // You can use $param1 and $param2 in your controller logic.
      return "Parameter 1: $param1, Parameter 2: $param2";
   }
}
```

3) How to pass variable which can be null in Route?

Ans:-

- ## *Define the Route with an Optional Parameter:*

In your '`routes/web.php`' file, define a route with an optional parameter:

```
Route::get('/example/{param?}', 'ExampleController@index');
```

In this example, we have defined a route with an optional parameter named '`param`'. The '`?`' makes it optional, and we've given it a default value of '`null`'.

- ## *Create a Controller:*

You should create a controller (if you haven't already) to handle this route. You can generate a controller using the '`artisan`' command-line tool:

php artisan make:controller ExampleController

This will create an '`ExampleController.php`' file in the '`app/Http/Controllers`' directory.

- ## *Handle the Optional Parameter in the Controller:*

In the '`ExampleController`', define the '`index`' method and capture the optional parameter. You can check if it's '`null`' or has a value:

```
namespace App\Http\Controllers;

use Illuminate\Http\Request;

class ExampleController extends Controller
{
   public function index($param = null)
   {
      if ($param === null) {
         return "Parameter is null";
      } else {
         return "Parameter is: $param";
      }
   }
}
```

4) Create custom auto using middleware.

Ans:-

- ## *Create the Custom Middleware:*

In your Laravel project, you can create custom middleware using the '`artisan`' command-line tool. Open your terminal and run the following command to generate a middleware:

php artisan make:middleware CustomAutoMiddleware

This command will create a new middleware file in the '`app/Http/Middleware`' directory, named '`CustomAutoMiddleware.php`'.

- ## *Define the Middleware Logic:*

Open the '**`CustomAutoMiddleware.php`**' file in a text editor or your IDE. In this middleware, you can define your custom logic. For this example, let's create a middleware that checks if a request contains a specific header to simulate an "auto" condition:

```
<?php

namespace App\Http\Middleware;

use Closure;

class CustomAutoMiddleware
{
    public function handle($request, Closure $next)
    {
        // Check if the request contains a specific header
        if ($request->header('X-Auto') === 'true') {
            // Perform auto-related actions
            // For example, add a response header
            $response = $next($request);
            $response->header('X-Auto-Response', 'Auto Response');
            return $response;
        }

        return $next($request);
    }
}
```

In this example, the middleware checks if the request contains the '**`X-Auto`**' header with a value of 'true'. If it does, it adds an '**`X-Auto-Response'`** header to the response.

- ## *Register the Middleware:*

To make the middleware available, you need to register it in the '**`app/Http/Kernel.php`**' file. Open the '**`Kernel.php`**' file and add your middleware to the '**`$middleware`**' or '**`$routeMiddleware`**' property as needed. For this example, add it to the '**`$routeMiddleware`**' property:

```
protected $routeMiddleware = [
    // ...
    'customAuto' => \App\Http\Middleware\CustomAutoMiddleware::class,
];
```

- ***Apply the Middleware:***

You can apply the custom middleware to routes, route groups, or controllers. Here's how to apply it to a route:

Route::get('/example', 'ExampleController@index')->middleware('customAuto');
In this example, the '**`customAuto`**' middleware will be applied to the '**`/example`**' route. Now, when you access the route '**`/example`**' with the '**`X-Auto: true`**' header in the request, the middleware will add the '**`X-Auto-Response'`** header to the response, simulating the auto-related functionality.

5) Generate Resource Controller for employee.
Ans:-

- ***Generate the Resource Controller:***

Create an EmployeeController with resourceful methods using the following command:

Php artisan make:controller employeeController --resource

main :- Php artisan make:controller employeeController --resource --model=Employee

This will create an "EmployeeController" in the '**`app/Http/Controllers`**' directory with predefined methods for CRUD operations (index, create, store, show, edit, update, destroy).

- ***Define Routes:***

In your '**`routes/web.php`**' file, define the routes for the '**`EmployeeController`**' using the '**`Route::resource`**' method:

use App\Http\Controllers\EmployeeController;

Route::resource('employees', EmployeeController::class);

This will automatically generate the necessary routes for CRUD operations on employees.

6) In Employee Controller"s action Call Middleware.

Ans:-

- ### *Method 1: Apply Middleware to a Specific Controller Method*

You can apply middleware to a specific controller method by using the '`middleware`' method within the controller. In this example, we'll apply middleware to the '`show`' method of the EmployeeController:

```
namespace App\Http\Controllers;

use Illuminate\Http\Request;

class EmployeeController extends Controller
{
   public function show(Request $request, $id)
   {
      // This middleware will only run for the "show" method.
   }
}
```

To apply middleware to the '`show`' method, add the '`middleware`' method within the controller like this:

```
public function show(Request $request, $id)
{
   $this->middleware('your-middleware-name');
   // Your method logic here
}
```

Replace `'your-middleware-name'` with the name of the middleware you want to apply.

- ### *Method 2: Apply Middleware to the Entire Controller*

You can apply middleware to the entire controller by defining it in the controller's constructor. This middleware will be applied to all methods within the controller:

```
namespace App\Http\Controllers;

 use Illuminate\Http\Request;

class EmployeeController extends Controller

{
```

```
    public function __construct()
    {
        $this->middleware('your-middleware-name');
    }
  // Other controller methods here
}
```

In this case, the middleware specified in the constructor will run for all methods within the '`EmployeeController`' unless overridden within specific methods.

Replace `'your-middleware-name'` with the name of the middleware you want to apply.

Make sure to register your middleware in the '`app/Http/Kernel.php`' file within the '`$routeMiddleware`' array.

```
protected $routeMiddleware = [
    'your-middleware-name' => \App\Http\Middleware\YourMiddleware::class,
];
```

Replace `'your-middleware-name'` with the actual name of your middleware class. This way, Laravel knows which middleware to run when you specify it in your controller.

7) How to remove route caching?

Ans:-

Open your terminal or command prompt.
Run the following Artisan command:

php artisan route:clear

Laravel will clear the cached routes, and you will see a message indicating that the route cache has been cleared.

Route cache cleared!

Now, your route cache has been removed, and Laravel will recompile your routes on the next request. This ensures that any changes you've made to your routes are immediately reflected in your application.

8) Create Custom Macro For search User

Ans:-

- ### *Create a MacroServiceProvider:*

You should create a new service provider to define your custom macro. Run the following command to generate a new service provider:

php artisan make:provider UserSearchMacroProvider

- ### *Edit the MacroServiceProvider:*

Open the 'UserSearchMacroProvider' you just created, which can be found in the 'app/Providers' directory, and define your custom macro in the 'boot' method:

```
<?php
namespace App\Providers;
use Illuminate\Support\ServiceProvider;
use App\User; // Replace 'App\User' with the actual User model namespace
class UserSearchMacroProvider extends ServiceProvider
{
  public function boot()
  {
    User::macro('search', function ($keyword) {
      return $this->where('name', 'like', '%' . $keyword . '%')
          ->orWhere('email', 'like', '%' . $keyword . '%');
    });
  }
  public function register()
  {
```

```
        // ...
    }
}
```

In the code above, we've created a 'search' macro for the 'User' model. It searches for users whose name or email matches a given keyword.

- ***Register the MacroServiceProvider:***

Add your newly created 'UserSearchMacroProvider' to the providers array in the 'config/app.php' file:

```
'providers' => [
    // ...
    App\Providers\UserSearchMacroProvider::class,
]
```

- ***Using the Custom Macro:***

You can now use the 'search' macro on your User model to search for users based on a keyword. For example:

```
$users = User::search('John')->get();
```