Contents

1 Explain MVC of larval

MVC is an acronym for 'Model View Controller'. It represents architecture developers adopt when building applications. With the MVC architecture, we look at the application structure with regards to how the data flow of our application works
MVC is a software architecture…that separates domain/application/business…logic from the rest of the user interface.
It does this by separating the application into three parts:
The model,
The view, and
The controller.
The model manages fundamental behaviors and data of the application. It can respond to requests for information, respond to instructions to change the state of its information, and even notify observers in event-driven systems when information changes. This could be a database or any number of data structures or storage systems. In short, it is the data and data-management of the application.
The view effectively provides the user interface element of the application. It'll render data from the model into a form that is suitable for the user interface.
The controller receives user input and makes calls to model objects and the view to perform appropriate actions.

A Model is a representation of a real-life instance or object in our code base. The View represents the interface through which the user interacts with our application. When a user takes an action, the Controller handles the action and

when you build using the MVC architecture, you have the following strategic advantages:

- **Splitting roles in your project are easier**.
  When the MVC architecture is adopted, you have the advantage of splitting roles in the project. You can have a backend developer working on the controller logic, while a frontend developer works on the views. This is a very common way to work in companies and having MVC makes it much easier than when the codebase has spaghetti code.
- **Structurally 'a-okay'**.
  MVC can force you to split your files into logical directories which makes it easier to find files when working on large projects.
- **Responsibility isolation**.
  When you adopt MVC, each broad responsibility is isolated. For instance, you can make changes in the views and the models separately because the model does not depend on the views.
- **Full control of application URLs.**
  With MVC architecture, you have full control over how your application appears to the world by choosing the application routes. This comes in handy when you are trying to improve your application for SEO purposes.
- **Writing SOLID code is easier**.
  With MVC it is easier to follow the SOLID principle.

2 Explain routing

Routing in Laravel allows users to route all their application demands to its appropriate controller. Most primary routes in Laravel acknowledge and accept a Uniform Asset Identifier together with a closure, giving a simple and expressive way of routing.

The route is a way of creating a request URL for your application. These URLs do not have to map to specific files on a website, and are both human readable and SEO friendly.

In Laravel, routes are created inside the routes folder. They are are created in the web.php file for websites. And for APIs, they are created inside api.php.

These routes are assigned to the net middleware group, highlighting the session state and CSRF security. The routes in routes/api.php are stateless and are assigned the API middleware group.
The default installation of Laravel comes with two routes, one for the web and the other for API. Here is how the route for web in web.php looks like:

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

return view('welcome');

});
```

All Laravel routes are defined in route files found within the routes directory. The application's App\Providers\RouteServiceProvider automatically stacks these files. And the routes/web.php file defines the routes for your web interface.

You can define a route to this controller action, as:

Route::get('user/{id}', 'UserController@show');

Route::resource: The Route::resource method can be a Restful Controller that produces all the basic routes required for an application and is dealt via controller class.

When a request matches the specified route URI, the show method on the App\Http\ControllersUserController lesson is invoked, passing the route parameters to the method.

For resources, you have to do two things on the Laravel application. Firstly, you must create a resource route on Laravel that provides insert, update, view, and delete routes. Secondly, create a resource controller that provides a method for insert, update, view, and delete.

The default installation of Laravel comes with two routes: one for the web and the other for API. Here is whatthe route for web in web.php looks like:

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

return view('welcome');

});
```

Laravel Middleware acts as a bridge between the request and the reaction. It can be a sort of filtering component.

Laravel joins a middleware that confirms whether or not the client's application is verified. In case the client is confirmed, it redirects to the home page or a login page.

Stop Wasting Time on Servers

The above code defines a route for the homepage. Every time this route receives a get request for /, It will return the view welcome. Views are the frontend of the application and I will discuss the topic in an upcoming installment of this series.

All Laravel routes are defined in your route files, which are found within the routes directory. Your application's AppProvidersRouteServiceProvider consequently stacks these records. The routes/web.php record defines routes that are for your web interface.

The structure of the route is very simple. Open the appropriate file (either `web.php` or `api.php`) and start the code with `Route::` This is followed by the request you want to assign to that specific route and then comes the function that will be executed as a result of the request.

Laravel offers the following route methods:

- get
- post
- put
- delete
- patch
- options

Routes are defined in Laravel within the Route class with an HTTP verb, the route to reply to, and closure, or a controller strategy.

How to Create Routes in Laravel

create your own routes in Laravel.

All the post, put, and delete requests require the CSRF token to be sent along with the request. Otherwise, the request will be rejected. Learn more about CSRF protection in Laravel official docs.

A Basic GET Route

I will now create a basic route that will print the table of 2.

```
Route::get('/table', function () {

for($i =1; $i <= 10 ; $i++){

echo "$i * 2 = ". $i*2 ."<br>";

}

});
```

In the above code, I created a GET request route for /table, which will print the table of 2 on the screen.

Now, what if I want the user to decide the number for which the route will print the table. Here is the code for such a route:

```
Route::get('/table/{number}', function ($number) {

for($i =1; $i <= 10 ; $i++){

echo "$i * $number = ". $i* $number ."<br>";

}

});
```

I have defined a route parameter in the above code and passed it to the route function. Now, whenever a user route to /table and add a number to the request (for instance, /table/3), the function will print the table of that number to the screen.

Right now, I have created two separate routes (one that outputs 2'stable and the other that displays table for a user-defined number).

But is there any way of combining both functionalities in a single route?

Yes, there is one. Laravel offers the functionality of Optional Parameters that lets you add optional parameters by adding ? after the optional parameter and the default value. Let's do this

```
Route::get('/table/{number?}', function ($number = 2) {

for($i =1; $i <= 10 ; $i++){

echo "$i * $number = ". $i* $number ."<br>";

}

});
```

In the above code we have made our route parameter, making the number optional, so if a user routes `/table` then it will generate 2's table by default and if a user routes to `/table/{number}` then the table of that number will be printed.

Regular Expressions Constraints For Route Parameters

Above, we have created a route for table, but how do we ensure if the route parameter is actually a number to avoid errors while table generation?

In Laravel, you can define a constraint for your route parameter by using the `where` method on route instance. The `where` method takes parameter name and a regular expression rule for that parameter.

Let's now define a constraint for our `{number}` parameter to assure that only a number is passed to the function.

```
Route::get('/table/{number?}', function ($number = 2) {

for($i =1; $i <= 10 ; $i++){

echo "$i * $number = ". $i* $number ."<br>";

}

})->where('number', '[0-9]+');
```

In the above code snippet, I defined a regular expression for our route's number. Now if a user try to route to /table/no, a NotFoundHttpException will be displayed.

Laravel Routing with Controller Function

In Laravel, you can define a Controller method to a route. A controller method performs all the defined actions whenever a user comes to the route.
Use the following code snippet to assign a controller method to a route:

```
Route::get('/home', 'YourController@functionname');
```

The code starts with `Route::` and then defines the request method for the route. Next, define your route and the Controller along with the method by adding the @ symbol before the method's name.

Naming Your Route

In Laravel, you can define name to your route. This name is often very useful in various scenarios. For example, if you want to redirect a user from one route to another, you do not need to define the full redirect URL. Instead, you can just give its name, and the code will work! You can define the route's name by using the `name` method in the route instance.

```
Route::get('/table/{number?}', function ($number = 2) {

for($i =1; $i <= 10 ; $i++){

echo "$i * $number = ". $i* $number ."<br>";

}

})->where('number', '[0-9]+')->name('table');
```

Now, I could re-generate the URL for this route through:

```
$url = route('table');
```

Similarly, for redirecting to this URL, the proper syntax would be:

```
return redirect()->route('table');
```

3 explain migration and relation

Migrations are like version control for your database, allowing your team to define and share the application's database schema definition. If you have ever had to tell a teammate to manually add a column to their local database schema after pulling in your changes from source control, you've faced the problem that database migrations solve.

❖ To generate a database migration.

make:migration Artisan command

A migration class contains two methods: up and down The up method is used to add new tables, columns, or indexes to your database, while the down method should reverse the operations performed by the up method. Within both of these methods, you may use the Laravel schema builder to expressively create and modify tables. To learn about all of the methods available on the schema builder, check out its documentation.

❖ To run all of your outstanding migrations, execute the migrate Artisan command:

php artisan migrate

- ❖ If you would like to see which migrations have run thus far, you may use the migrate:status Artisan command:

php artisan migrate:status

- ❖ If you would like to see the SQL statements that will be executed by the migrations without actually running them, you may provide the –pretend flag to the migrate command:

php artisan migrate –pretend

- ❖ isolatimg migration execution

php artisan migrate –isolated

Defining relationship

Eloquent relationships are defined as methods on your Eloquent model classes. Since relationships also serve as powerful query builder. defining relationships as methods provides powerful method chaining and querying capabilities

Database tables are often related to one another.

- ❖ One to one
- ❖ One to many
- ❖ Many to many
- ❖ Has one through
- ❖ Has many through
- ❖ One to one (polymorphic)
- ❖ One to many (polymorphic)
- ❖ Many to many (polymorphic)

4 Explain Blade template engine

Laravel Blade template engine enables the developer to produce HTML based sleek designs and themes.

All views in Laravel are usually built in the blade template. Blade engine is fast in rendering views because it caches the view until they are modified. All the files in **resources/views** have the extension **.blade.php.**

Laravel blade template is pretty impressive in performance. This tutorial will serve as an introduction to this templating engine.

5 Directives

directives are shortcut codes for the implementation of basic PHP structure control, such as loop and conditional statements. It makes your code snippets clean and easy to understand.

Note: These directives are basically only used in the blade template, so don't try to use them on your controllers.

In this shot, we look at some important blade directives.

Please refer to this shot if you don't know the basics of blade templating.

The if blade directive

- @if()
- @elseif()
- @else
- @endif

The if(){} statement is similar to the endif statement, except that the endif statement will serve as the closing {} curly braces.

The if(){} statement can be used like so:

@if(1=1)

{{ one will always be one.lol }}

@endif

The Auth blade directive

@auth

@endauth

This blade directive is used to check if a particular user has been authenticated.

Refer to this shot for more information on this.

The foreach blade directive

@foreach()

@endforech

These are loop directives and they work like the normal foreach(){}, except this directive is cleaner.

The foreach blade directive can be used like so:

@foreach($users as $user)

{{ $user->name }}

@endforech

The csrf blade directive

The @csrf directive is used in the form.

It is also used in Laravel applications to verify tokens.

Please refer to this shot to learn more on csrf protection.

{{ }} this is the most used blade directive and is similar to the <?php echo $string; ?>. It is basically used to echo the values of variables. It is used like so: {{ date('Y') }}.

The unless blade directive

@unless (Auth::check())

  You are not signed in.

@endunless

The code snippet above does the opposite of the @if blade directives.

The isset and empty blade directives

@isset($records)

@endisset

@empty($records)

@endempty

The isset() and empty() blade directives work normally, as shown in the above code snippet.

@guest

@endguest

The above code snippet checks if the the current user is a guest, i.e., if the user is not authenticated.

The production blade directive

@production

@endproduction

There are times when you want to run a particular code while your application is on production. In the above example, your code will only execute if the application is in production.

The env blade directive

```
@env('staging')
    // The application is running in "staging"...
@endenv

@env(['staging', 'production'])

@endenv
```

You can choose what environment you want to use to execute any code with the above code snippet.

The hasSection blade directive

```
@hasSection('navigation')

@endif
```

The code above checks if a section inherited has content.

The sectionMissing blade directive

```
 @sectionMissing('navigation')

@endif
```

The above code checks if the section's contents are missing.

The switch blade directive

```
@switch( $i)
    @case(1)
        First case...
        @break
    @default

@endswitch
```

Just like your normal switch statement, the @switch() blade directive functions the same.

The for loop blade directive

```
@for ($i = 0; $i < 10; $i++)

   The current value is {{ $i }}

@endfor
```

The blade directives are just simple and short forms that Laravel uses to simplify some PHP functions to make your codes simple and readable.