

DEBERE BERHAN UNIVERSITY

COLLEGE of COMPUTING

DEP't Of Computer Science

Selected topics in computer science
Individual Assignment

Name : Anteneh Sahle

ID: DBUE/769/11

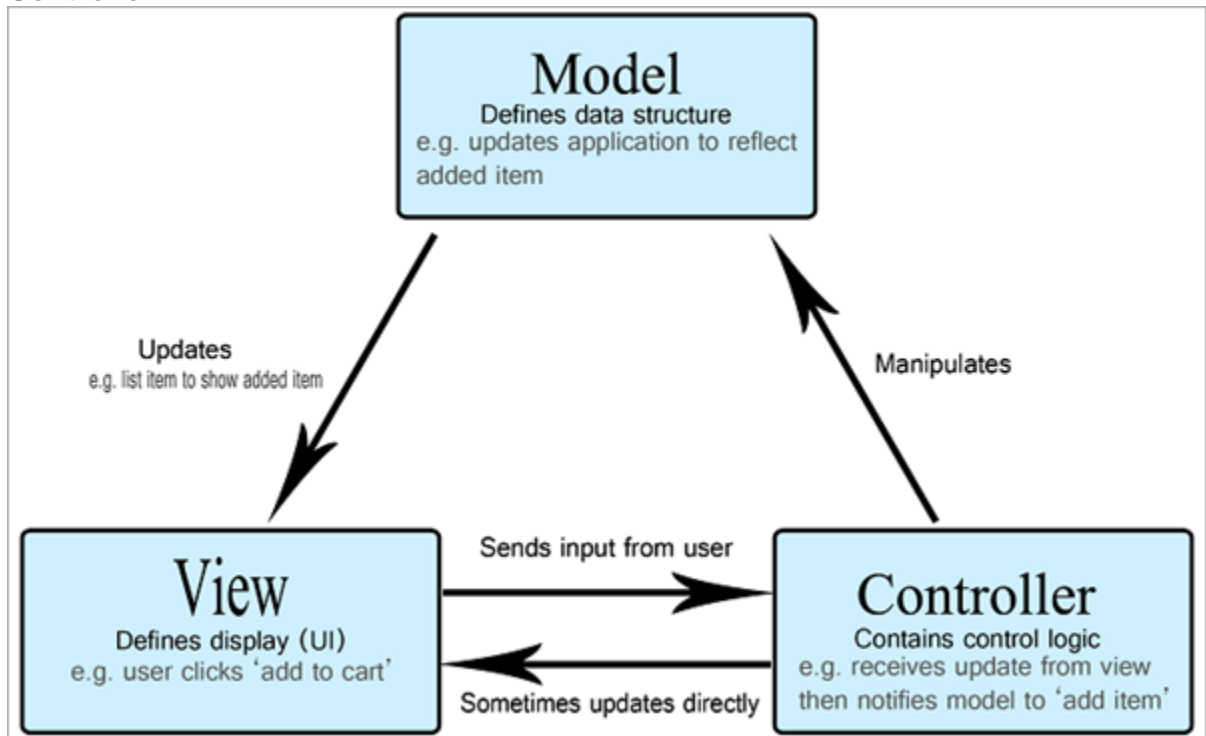
Sub. To Girmachew
Sub. Date Jan 28 /2023

1. Explain MVC of laravel

MVC is an architectural design pattern that helps to develop web applications faster.
MVC stands for **Model-View-Controller**.

- **Model (M)**—A model handles data used by the web application.
- **View (V)**—A view helps to display data to the user.
- **Controller (C)**—A controller interacts with the model to create data for the view.

The following screenshot shows the interactions between Model, View, and Controller.



2. Explain Routing

Routing is one of the essential concepts in Laravel. Routing in Laravel allows you to route all your application requests to their appropriate controller. The main and primary routes in Laravel acknowledge and accept a URI (Uniform Resource Identifier) along with a closure, given that it should have to be a simple and expressive way of routing. In this tutorial, you will learn about the routing concept of Laravel.

Create Routes in laravel

All the routes in Laravel are defined within the route files you can find in the routes sub-directory. These route files get loaded and generated automatically by the Laravel framework. The application's route file gets defined in the app/Http/routes.php file. The general routing in Laravel for each of the possible requests looks something like this:

http://localhost/

```
Route:: get ('/', function () {  
    return 'Welcome to index';  
});
```

http://localhost/user/dashboard

```
Route:: post('user/dashboard', function () {  
    return 'Welcome to dashboard';  
});
```

http://localhost/user/add

```
Route:: put('user/add', function () {  
    //  
});
```

http://localhost/post/example

```
Route:: delete('post/example', function () {  
    //  
});
```

The routing mechanism in laravel

The routing mechanism takes place in three different steps:

1. First of all, you have to create and run the root URL of your project.
2. The URL you run needs to be matched exactly with your method defined in the root.php file, and it will execute all related functions.
3. The function invokes the template files. It then calls the `view()` function with the file name located in `resources/views/`, and eliminates the file extension `blade.php` at the time of calling.

3.Explain Migration and Relationships

Laravel Migration

What is Laravel Migration?

Laravel Migration is an essential feature in Laravel that allows you to create a table in your database. It allows you to modify and share the application's database schema. You can modify the table by adding a new column or deleting an existing column.

Why do we need Laravel Migration?

Suppose we are working in a team, and some idea strikes that require the alteration in a table. In such a case, the SQL file needs to be passed around, and some team member has to import that file, but team member forgot to import the SQL file. In this case, the application will not work properly, to avoid such situation, Laravel Migration comes into existence.

Laravel Migration allows you to add a new column or delete the records in your database without deleting the records that are already present.

Laravel Relationship

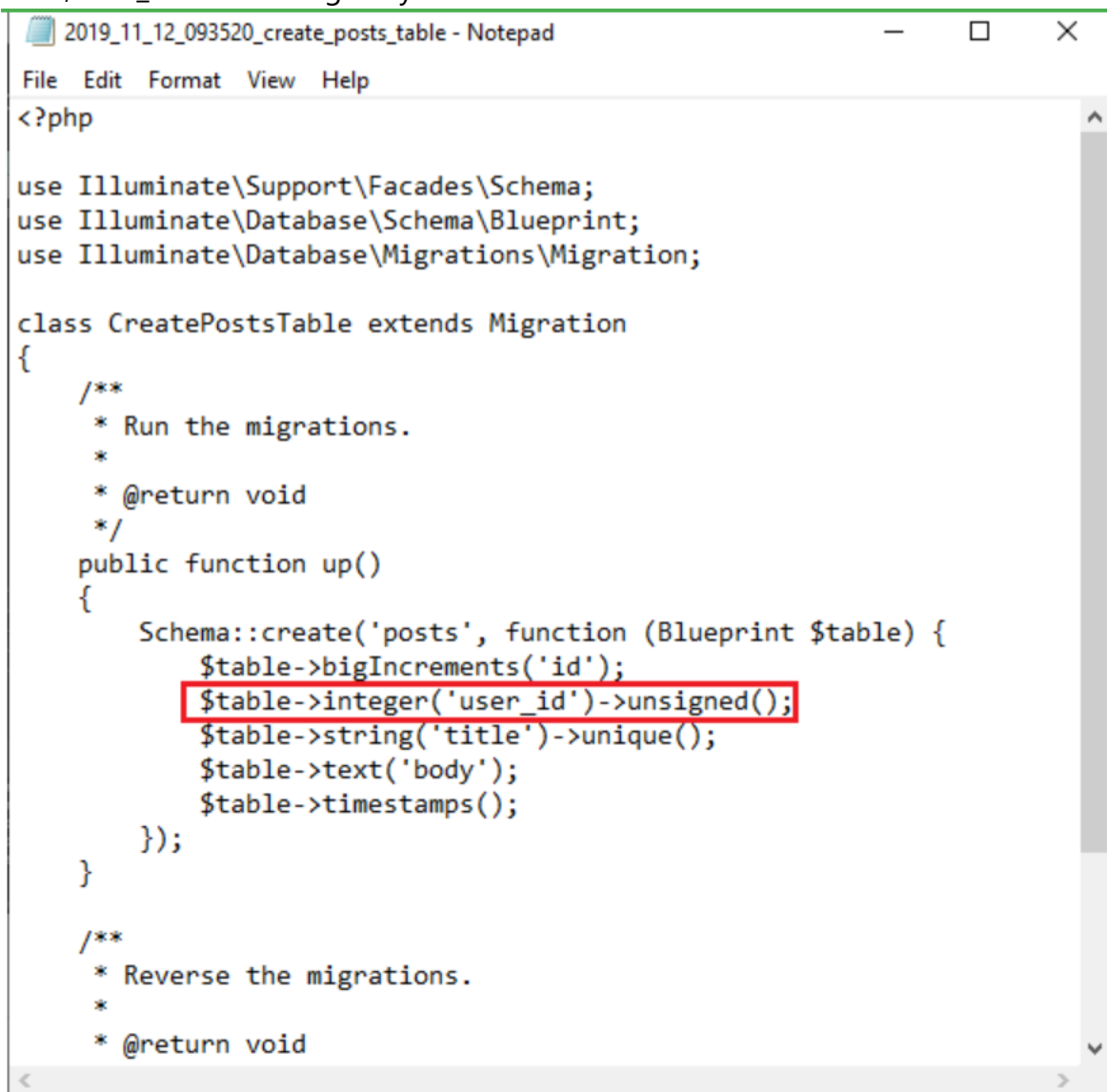
An **Eloquent relationship** is a very important feature in Laravel that allows you to relate the tables in a very easy format.

One to one relationship

One to one relationship provides the one-to-one relationship between the columns of different tables. For example, every user is associated with a single post or maybe multiple posts, but in this relationship, we will retrieve the single post of a user. To define a relationship, we need first to define the **post()** method in User model. In the post() method, we need to implement the **hasOne()** method that returns the result.

Let's understand the one to one relationship through an example.

- First, we add the new column (user_id) in an existing table named as **posts**. Here, user_id is the foreign key



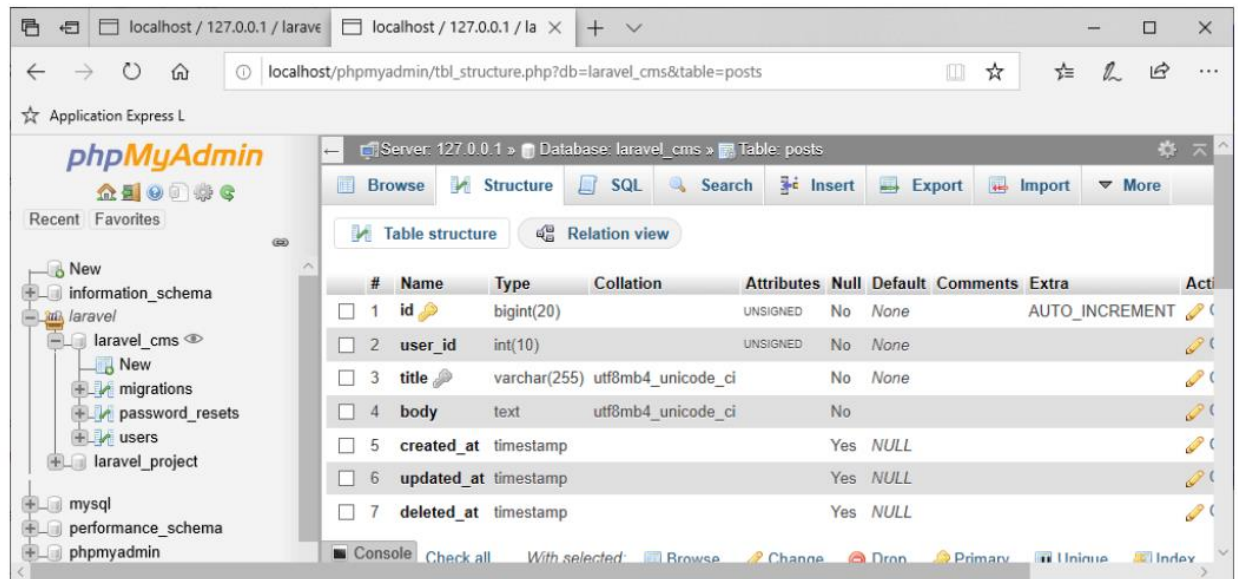
```
2019_11_12_093520_create_posts_table - Notepad
File Edit Format View Help
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreatePostsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('posts', function (Blueprint $table) {
            $table->bigIncrements('id');
            $table->integer('user_id')->unsigned();
            $table->string('title')->unique();
            $table->text('body');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
}
```

- Migrate the above changes in a database by using the command given below:
php artisan migrate.
- After the migration, the structure of the **posts** table is shown in the below screenshot:



The above screenshot shows that the **user_id** column is added successfully.

One-to-many relationship

Laravel also provides a one-to-many relationship.

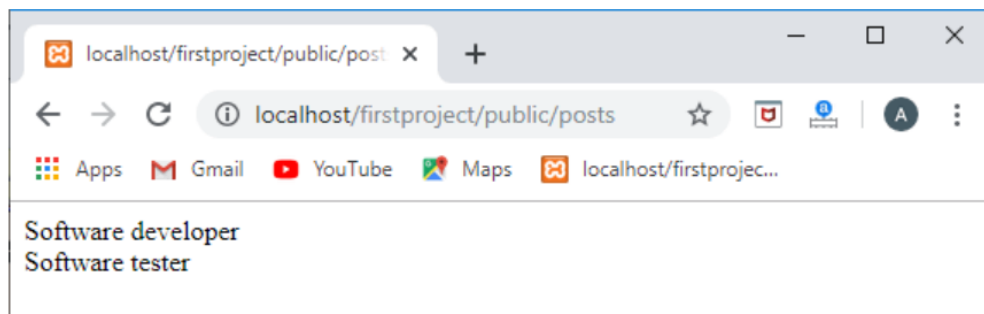
- First, we define the route that finds out all the posts of a single user.

```
Route::get('/posts',function(){
    $user=User::find(1);
    foreach($user->posts as $post){
        echo $post->title."<br>";
    }
});
```

- Add the following code in the **User.php(model)** file.

```
public function posts()
{
    return $this->hasMany('App\Post','user_id');
}
```

Output

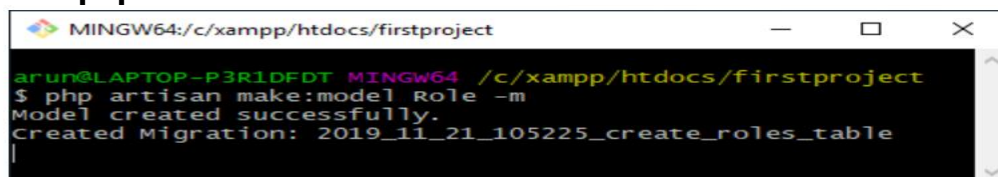


Many-to-Many relationship

A many-to-many relationship is more complicated than a one-to-one relationship and a one-to-many relationship. To define the many-to-many relationship, we need to create a pivot table. The pivot table is basically a look up table that relates the two tables. For example, a user may have different roles where the roles can be shared by other users, like many users can have the role of '**Admin**'. To define the relationship between the users and roles, we need to create the three tables, user, roles, and role_user. In our database, the user table is already created; we need to create two tables, i.e., roles table and pivot table (roles_user).

- First, we create the **Role** model. We use the following command to create the model:

php artisan make:model Role -m



4. 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.

Create Pages Routes

First, I will create the routes to gain access to my page.

Open routes/web.php and add the following lines in it.

```
1. Route::get('/', function()
2.
3. {
4.
5.     return View::make('pages.home');
6.
7. });
8.
9. Route::get('/about', function()
10.
11. {
12.
13.     return View::make('pages.contact');
14.
15. });
```


Create Views Structure

Now that the routes are ready, let's create the Views structure by creating the following folders and files.

```
1.  - resources
2.
3.  -- views
4.
5.  --- layouts
6.
7.  ----- default.blade.php
8.
9.  --- pages
10.
11. ----- home.blade.php
12.
13. ----- contact.blade.php
14.
15. --- includes
16.
17. ----- head.blade.php
18.
19. ----- header.blade.php
20.
21. ----- footer.blade.php
```

Create Includes

Create the following includes, with the following code:

head.blade.php

```
1.  <meta charset="utf-8">
2.
3.  <meta name="description" content="">
4.
5.  <meta name="Saquib" content="Blade">
6.
7.  <title>Checkout our layout</title>
8.
9.  <!-- load bootstrap from a cdn -->
10.
11. <link rel="stylesheet" href="//netdna.bootstrapcdn.com/twitter-bootstrap/3.0.3/css/bootstrap-combined.min.css">
```

Header.blade.php

```
1. <div class="navbar">
2.
3.     <div class="navbar-inner">
4.
5.         <a id="logo" href="/">Single Malt</a>
6.
7.         <ul class="nav">
8.
9.             <li><a href="/">Home</a></li>
10.
11.             <li><a href="/contact">Contact</a></li>
12.
13.         </ul>
14.
15.     </div>
16.
17. </div>
```

footer.blade.php

```
1. <div id="copyright text-right">© Copyright 2017 Saquib Rizwan </div>
```

Conclusion

This is the simple foundation and the basic steps for getting starting with Blade templating engine. Now that you know the file structures and the way Laravel Blade template engine renders the view, I will demonstrate how to integrate Bootstrap template with Laravel in my next article. So stay tuned.

5. Directive

Blade 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.

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 **Auth** blade directive

- @auth
- @endauth

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

The **foreach** blade directive

- @foreach()
- @endforeach

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

The csrf blade directive

- The `@csrf` directive is used in the `form`.
- It is also used in Laravel applications to verify tokens.

References

- ❖ <https://www.educative.io/answers/what-are-blade-directives>
- ❖ <https://www.cloudways.com/blog/create-laravel-blade-layout/>
- ❖ <https://www.javatpoint.com/laravel-migration#:~:text=What%20is%20Laravel%20Migration%3F,or%20deleting%20an%20existing%20column.>