ደብረ ብርሃን ዩኒቨርሲቲ

D B U

DEBRE BERHAN UNIVERSITY

COLLEGE OF COMPUTING

DEPARTMENT: COMPUTER SCIENCE

COURSE TITLE: SELECTED TOPIC IN COMPUTER SCIENCE

COURSE CODE: COSC4181

INDIVIDUAL ASSIGNMENT

BY:MENGESHA AREGAHEGN ID: DBUE/796/11

SUBMITTED TO GIRMACHEW G.

January 28/2023

Q1. Explain MVC of laravel

- Laravel uses the MVC model; therefore there are three core-parts of the framework which work together: models, views and controllers.

Model

- It represents an individual database table or a record from that table

View

- It repents the template that output your data to the end user think the login page template with tis given set of HTML and CSS and also JavaScript.

Controller

- Controllers are the main part where most of the work is done. Like a traffic cop, takes HTTP request from the browser, gets the right data out of the database and other storage mechanism, validate user input, and eventually sends a response back to the user
- The end user will first interact with the controller by sending an HTTP request using their browser. The controller response to that request, may write data to and/or pull data from the model (database).the controller will the likely send data to a view and then a view will be returned the end user to display in the browser

Q2. Explain Routing

- Routing defines a map between HTTP methods and URIs on one side, and actions on the other.
- Routes are normally written in the app/Http/routes.php file.
- In its simplest form, a route is defined by calling the corresponding HTTP method on the Route facade, passing as parameters a string that matches the URI (relative to the application root), and a callback.

For instance: a route to the root URI of the site that returns a view home looks like this:

```
Route::get ('/', function () {
Return view ('home');
  });
```

- Instead of defining the callback inline, the route can refer to a controller method in [ControllerClassName@Method]

```
Route::get('login', 'LoginController@index');
```

- The match method can be used to match an array of HTTP methods for a given route:

```
Route::match(['GET', 'POST'], '/', 'LoginController@index');
```

- Also you can use all to match any HTTP method for a given route:

```
Route::all('login', 'LoginController@index');
```

- Routes can be grouped to avoid code repetition.
- Let's say all URIs with a prefix of /admin use a certain middleware called admin and they all live in the App\Http\Controllers\Admin namespace.
- A clean way of representing this using Route Groups is as follows:

```
Route::group([
'namespace' => 'Admin',
'middleware' => 'admin',
'prefix' => 'admin'], function () {
Route::get('/', ['uses' => 'IndexController@index']);
Route::get('/logs', ['uses' => 'LogsController@index']);
        });
```

Q3. Explain migration and relationship

- Modern framework like laravel make it easy to define your database structure with code driven migration .every new table , column ,index and key can be defined in code , and any new environment can be brought from bare database to your app's schema in seconds
- A migration is a single file that define two things :the modifications desired when routing this migration up and optionally, the modification desired when running this migration down
- One migration file should represent a schema update to solve a particular problem.
- Database migrations are kept in chronological order so that Laravel knows in which order to execute them. Laravel will always execute migrations from oldest to newest.
- Creating a new migration file with the correct filename every time you need to change your schema would be a chore. Thankfully, Laravel's artisan command can generate the migration foryou.

```
php artisan make:migration users_table
```

- You can also use the --table and --create flags with the above command. These are optional and just for convenience, and will insert the relevant boilerplate code into the migration file.
- Each migration should have an up() method and a down() method. The purpose of the up() method is to perform the required operations to put the database schema in its new state, and the purpose of the down() method is to reverse any operations performed by the up() method. Ensuring that the down() method correctly reverses your operations is critical to being able to rollback database schema changes.

An example migration file may look like this:

```
<?php
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;
class AddLastLoggedInToUsersTable extends Migration{
public function up(){
Schema::table('users', function (Blueprint $table) {
$table->dateTime('last_logged_in')->nullable();
});}
public function down() {
Schema::table('users', function (Blueprint $table) {
$table->dropColumn('last_logged_in');
});
```

Relationship

Eloquent relationships are defined as functions on your eloquent model classes. Since, like Eloquent models themselves, relationships also serve as powerful query builders, defining relationships as functions provides powerful method chaining and querying capabilities

Relationship Types

One to Many

Lets say that each Post may have one or many comments and each comment belongs to just a single Post. so the comments table will be having post_id. In this case the relationships will be as follows.

Post Model

```
public function comments() {
return $this->belongsTo(Post::class);
}
```

If the foreign key is other than post_id, for example the foreign key is example_post_id.

```
public function comments()
{
return $this->belongsTo(Post::class, 'example_post_id');
}
```

and plus, if the local key is other than id, for example the local key is other_id

```
public function comments()
{
return $this->belongsTo(Post::class, 'example_post_id', 'other_id');
}
}
```

Comment Model

defining inverse of one to many

```
public function post()
{
return $this->hasMany(Comment::class);
}
```

<u>One to One</u>

- How to associate between two models (example: User and Phone model)

<u>Many to Many</u>

- Lets say there is roles and permissions. Each role may belongs to many permissions and each permission may belongs to many role. so there will be 3 tables. two models and one pivot table. A roles, users and permission_role table.

Q4. Explain blade template engine

- Laravel supports Blade templating engine out of the box. The Blade templating engine allows us to create master templates and child templating loading content from master templates; we can have variables, loops and conditional statements inside the blade file.
- It is shipped with a templating engine known as Blade. Blade is quite easy to use, yet,powerful. One feature the Blade templating engine does not share with other popular ones is her permissiveness; allowing the use of plain PHP code in Blade templating engine files.
- It is important to note that Blade templating engine files have .blade appended to file names right before the usual .php which is nothing other than the actual file extension. As such, .blade.php is the resulting file extension for Blade template files. Blade template engine files are stored in the resources/views directory.

NAMING CONVENTION FOR BLADE VARIABLES

While sending data back to view. You can use underscore for multi-words variable but with – laravel gives error.

Like this one will give error (notice hyphen (-) within the user-address

```
view('example',['user-address' => 'Some Address']);
```

The correct way of doing this will be

```
view('example', ['user_address' => 'Some Address']);
```

Control Structures

Blade provides convenient syntax for common PHP control structures.

Each of the control structures begins with @[structure] and ends with @[endstructure].

Notice that within the tags, we are just typing normal HTML and including variables with the Blade syntax.

Conditionals

'If' statements

```
@if ($i > 10)
<p>{{ $i }} is large.</p>
@elseif ($i == 10)
<p>{{ $i }} is ten.</p>
@else
<p>{{ $i }} is small.</p>
@endif
```

'Unless' statements

```
@unless ($user->hasName())
<p>A user has no name.</p>
@endunless
```

Loops

While loop

```
@while (true)
<p>I'm looping forever.</p>
@endwhile
```

Foreach loop

```
@foreach ($users as $id => $name)
<p>User {{ $name }} has ID {{ $id }}.</p>
@endforeach
```

Forelse Loop

Same as 'foreach' loop, but adds a special @empty directive, which is executed when the array

Expression iterated over is empty, as a way to show default content.

```
@forelse($posts as $post)
<p>{{ $post }} is the post content.</p>
@empty
<p>There are no posts.</p>
@endforelse
```

Q4. Directive

- In addition to template inheritance and displaying data, Blade also provides convenient shortcuts for common PHP control structures, such as conditional statements and loops. These shortcuts provide a very clean, terse way of working with PHP control structures while also remaining familiar to their PHP counterparts.

If Statements

You may construct if statements using the @if, @elseif, @else, and @endif directives. These directives function identically to their PHP counterparts:

```
@if (count($records) === 1)
I have one record!
@elseif (count($records) > 1)
I have multiple records!
@else
I don't have any records!
@endif
```

For convenience, Blade also provides an @unless directive:

```
@unless (Auth::check())
You are not signed in.
@endunless
```

In addition to the conditional directives already discussed, the @isset and @empty directives may be used as convenient shortcuts for their respective PHP functions:

```
@isset($records)
// $records is defined and is not null...
@endisset
```

Authentication Directives

The @auth and @guest directives may be used to quickly determine if the current user is authenticated or is a guest:

```
@auth
// The user is authenticated...
@endauth
@guest
// The user is not authenticated...
@endguest
```

Environment Directives

You may check if the application is running in the production environment using the @production directive:

```
@production
// Production specific content...
@endproduction
```