COLLEGE OF COMPUTING.

DEPARTEMENT OF COMUTER SCIENISE.

SUBJECT- SELECTED TOPICS IN CS.

PREPARED BY- GETENET T/YOHANNES

ID DBUE/786/11

1.What is MVC?

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

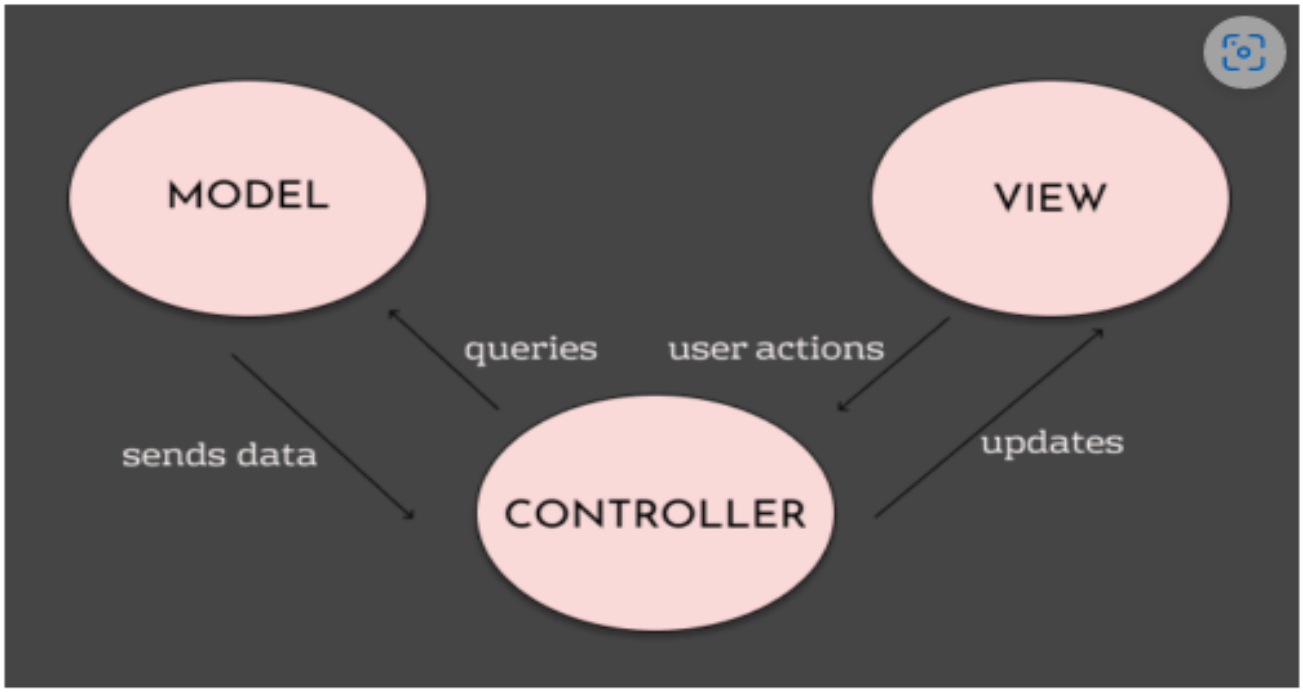
MVC is a software architecture…that separates domain/application/business…logic from the rest of the user interface. It does this by separating the application into three parts: the model, the view, and the controller.

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.

All in all, these three components work together to create the three basic components of MVC.
– **Bob, Stack Overflow**

We have a structure that looks like this:



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 updates the Model if necessary.

Let’s look at a simple scenario.

If you go to an e-commerce website, the different pages you see are provided by the View layer. When you click on a particular product to view more, the Controller layer processes the user’s action. This may involve getting data from a data source using the Model layer. The data is then bundled up together and arranged in a View layer and displayed to the user. Rinse and repeat.

Why use MVC?

This article is not a comparison between architecture types but information on a single type, which is MVC.

When building PHP applications, it may be okay to have a lot of files flying around in very very small projects. However, when the project becomes even slightly bigger than five files or entry points having a structure can drastically improve maintainability.

When you have to work with codebases that have no architecture, it will become extremely grueling, especially if the project is big and you have to deal with unstructured code laying everywhere. Using MVC can give your code some structure and make it easier to work with.

On a more technical note, 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. what is 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']);

});
```

3. what is migration and relationships

Laravel migration is a way that allows you to create a table in your database, without actually going to the database manager such as phpmyadmin or sql lite or whatever your manager is.

Why should you consider doing a Migration when you already have the ability to create the tables directly?

The whole point in doing Laravel Migrations is the ability to perform sort of a **version control** to your database. So, actually when you can revert back to previous version or deploy the next version for your tables why don't you consider doing migrations? you're probably like "pssst…Is it just that?" Ofcourse not!

Imagine this : You're working with a team & suddenly a new idea strikes, so you have to alter the table structure right now. What do you do?

Well, you probably pass around the .sql file around & that Joseph there was busy texting his gf that he forgot to import that .sql file & it so happened that the application was broken & you're pissed at him.

Actually by doing migrations, you can essentially tackle the team collaboration issues as concerned with databases.

Migrations allow you to add or drop fields in your database without deleting the records already present.

How is it possible?

Laravel keeps a track of which migrations have already been executed within the table. This way, it will only do new additions or deletions of columns based off of your requests.

Ok Cool. Let me give it a try, how should I get started?

Step-1: The first step is to first create a migration file. This command basically creates a file in which you can define the structure of the table you wanna create.

Do this in your command prompt:

```
php artisan make:migration  create_articles_table --create=articles
```

the “create” suggests that you need to create a table whose name is “articles”. “create_articles_table” is a name for the migration.

Step-2: Specify what is to be done when you run a migration.This usually would mean either adding a column or 2 or howsoever. In my tutorial, I wanna add 2 fields such as “name” and “content” for the article. Usually the migration file comes along with the “id” & “timestamps” fields by default.

Anything you want to execute during a migration goes inside the “up” method
While, anything that needs to be reverted after migration goes inside the “down” method.

What is relationships?

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 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

One-to-many (Polymorphic)

The Polymorphic relationship is similar to the one-to-many relationship. When a single model belongs to more than one type of model on a single association is known as one-to-one polymorphic relationship. For example, if we have three tables, posts, users, and photo table, where photo table represents the polymorphic relation with the users and posts table.

Many-to-many polymorphic relationship

In a many-to-many polymorphic relationship, a target model consists of unique records that are shared among the various models. For example, a tag table shares the polymorphic relation between the videos and the posts table. A tag table consists of the unique list of tags that are shared by both the tables, videos, and the posts table.

Inverse Relation

Inverse relation means the inverse of the one-to-one relationship. In the above, we have retrieved the post belonging to a particular user. Now, we retrieve the user information based on the post.

4.Blade Template Engine

Blade Template

The Blade is a powerful templating engine in a Laravel framework. The blade allows to use the templating engine easily, and it makes the syntax writing very simple. The blade templating engine provides its own structure such as conditional statements and loops. To create a blade template, you just need to create a view file and save it with a .blade.php extension instead of .php extension. The blade templates are stored in the /resources/view directory. The main advantage of using the blade template is that we can create the master template, which can be extended by other files.

Why Blade template?

Blade template is used because of the following reasons:

- ✓ **Displaying data**

If you want to print the value of a variable, then you can do so by simply enclosing the variable within the curly brackets.

Syntax

{{$variable}};

In blade template, we do not need to write the code between <?php echo $variable;

?>. The above syntax is equivalent to <?= $variable ?>.

- ✓ **Ternary operator**

In blade template, the syntax of ternary operator can be written as:

{{ $variable or 'default value'}}

The above syntax is equivalent to <?= isset($variable) ? $variable : ?default

value? ?>

- ✓ Blade template engine also provides the control statements in laravel as well as shortcuts for the control statements.
- ✓ Blade template provides @unless directive as a conditional statement. The above code is equivalent to the following code:
- ✓ The blade templating engine also provides the @hasSection directive that determines whether the specified section has any content or not.
- ✓ The blade templating engine provides loops such as @for, @endfor, @foreach, @endforeach, @while, and @endwhile directives. These directives are used to create the php loop equivalent statements.

5. Directives

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