WEB ENGINEERING

**LARAVEL**

Assignment 3

horizontal line

# 

Group BSCS 6A

* Waqas Yaseen 196819
* Arifullah Jan 186943
* Bilal Khalid 128608
* Saqib Shakeel 176783

# Eloquent ORM([link](https://laravel.com/docs/5.6/eloquent))

ORM stands for Object Relational Mapper. The ORM in laravel is called Eloquent ORM. Basically an Object Relational Mapper makes it easy for programmer to work with the relational databases by enabling him/her to work with a easy syntax. It makes it easy for the programmer to perform the simple repeating tasks.

Eloquent ORM also creates a schema for the database. It allows for creating rules and relations among multiple models. The relation types are

1. One-One
2. One-Many
3. Many-Many

## Defining Relation:

In Eloquent model relations among different models can be defined. To define a relation of the current model with another a method is added which tells what is the type of the relation.

**For example**

To define a schema in which the current model is related to multiple model instances one would define it as:

class Model1 extends Eloquent {

public function user\_profile()

{

return $this->**has\_many(Model2)**;

}

}

# Querying:

**Easy Retrieval of Record:**

With Eloquent ORM one can easily retrieve a record by find(id) function. What it does is that it takes id and queries the database for the record in the particular table.

**Student::find($sid)**

One can also use where(key,operator,value) to find a record where a key is the column name and the value is the value which is to be operated with the operator.The operator can be “=”,”<=”,”>=” etc. The result is returned as an array of the matched records.

The alternative is where\_key(value) which has the key in the method name itself. Here it is clear that it is a framework we are calling a function which is created by the framework and the name is specific to our schema.

**Insertion & Updation of Records:**

The insertion is just like creating a new object.

**$var = new TableName();**

Now it is time to update the values of the record. This again can be done by just changing the values of the object and then calling save() function of the object.

**$var -> key = “another value”;**

**$var -> save();**

It also has other features like orderby, chunk etc.

# Reverse routing

Laravel provides a good routing mechanism. It automatically calls the respective model and views etc which are responsible for the task. The mechanism is configured in **routes directory** in the project.

A basic example can be where different stateless controllers are mapped on the urls as shown below

**Route::get('/products', 'ProductsController@index');**

Here the Products Controller will be executed if the url is domain/products.

The routing mechanism gets complex for app where state matters. Laravel allows parameters while requesting. The parameters can then be used to determine the data to be loaded or some other behaviour.

The methods that can be used to configure routing are Get,Post,Put, Patch, Delete and Options. They all get **uri** and **callback function** as parameters.

Other features include regex support, which is very critical and useful, CSRF protection, redirecting, named routes etc

Named Routes allows for changing of the controller names without changing urls through out the app.

# Application logic

The default Laravel application structure is proposed to give an extraordinary beginning stage to both huge and little applications. Obviously, you are allowed to sort out your application anyway you like. Laravel forces no confinements on where any given class is found - as long as Composer can autoload the class.

The application index ships with an assortment of extra catalogs, for example, Console, Http, and Providers. Think about the Console and Http indexes as giving an API into the "center" of your application. The HTTP convention and CLI are the two instruments to interface with your application, yet don't really contain application rationale. As it were, they are basically two methods for issuing orders to your application. The Console index contains the majority of your Artisan charges, while the Http catalog contains your controllers, channels, and demands.

The Commands index, obviously, houses the orders for your application. Orders speak to employments that can be lined by your application, and additionally undertakings that you can run synchronously inside the present demand lifecycle.

The Events index, as you may expect, houses occasion classes. Obviously, utilizing classes to speak to occasions isn't required; in any case, on the off chance that you utilize them, this registry is the default area they will be made by the Artisan order line.

The Handlers index contains the handler classes for the two charges and occasions. Handlers get a charge or occasion and perform rationale because of that order or occasion being let go.

The Services index contains different "aide" benefits your application needs to work. For instance, the Registrar benefit included with Laravel is in charge of approving and making new clients of your application. Different illustrations may be administrations to connect with outer APIs, measurements frameworks, or even administrations that total information from your own application.

The Exceptions catalog contains your application's exemption handler and is likewise a decent place to stick any special cases tossed by your application.

# Restful controllers

# As opposed to describing most of your request managing basis as Closures in course archives, you may wish to orchestrate this lead using Controller classes. Controllers can total related request managing justification into a single class. Controllers are secured in the application/Http/Controllers file.

# 

# the controller expands the base controller class included with Laravel. The base class gives two or three solace procedures, for instance, the middleware methodology, which may be used to annex middleware to controller exercises.

# 

# It is imperative to observe that we didn't need to demonstrate the full controller namespace while describing the controller course. Since the RouteServiceProvider loads your course records inside a course suspect that contains the namespace, we just showed the section of the class name that comes after the App\Http\Controllers bit of the namespace.

# 

# Laravel resource coordinating names the normal "Refuse" courses to a controller with a lone line of code. For example, you may wish to make a controller that handles all HTTP requests "photos" set away by your application. Using the make:controller Artisan charge, we can quickly make such a controller.

# 

# 

# View composers

To explain view composers we first need to know what is views.Views provide a very easy way to separate out the controller and server logic from the presentation logic These contain html which the application serves.  
This provide solution to problem which was occuring many times to the developers when they get errors from many views of undefined variables so the developers has go through the controller constructors to check where was the error in fetching any of the view info but view composers fix this for us itself

Laravel provides an outstanding functionality to handle views i.e. if we want to bind data with view , every time the view is altered there are methods or call backs known as View Composer which are called and binds data to the views. Laravel provides flexibility for the directory option for view composers as it does not locate any default directly, the user is free to organize the directories as they wish.

Service container performs the job of resolving all view composers giving us flexibility to type hint any dependency inside the constructors of composer.We can attach a composer to multiple views at a time.  
We can also use composer method for the registration of many composers at a time



# Bundles/Packages

In laravel 5.6 packages are the way the functionality is divided. A package is just like a module in any other framework. The packages can be used in any other laravel app.

Service providers are used to link the package to Laravel. Service is responsible for informing Laravel about the loading of views etc.

The configuration of the package should be included in the app’s config file so that the users of the package can change it.

All other configurations like getting the routes, registering views, publishing file groups, public assets etc are done by calling funcitons.

# Blade templating engine

Bleeding edge is the fundamental, yet powerful templating engine outfitted with Laravel. Not at all like other standard PHP templating engines, Blade does not confine you from using plain PHP code in your viewpoints. Honestly, all Blade sees are requested into plain PHP code and held until the point that they are balanced, which implies Blade adds essentially zero overhead to your application. Bleeding edge see archives use the .blade.php record extension and are regularly secured in the advantages/sees list.

Despite design inheritance and indicating data, Blade moreover gives worthwhile simple courses to ordinary PHP control structures, for instance, prohibitive clarifications and circles. These simple courses give an impeccable, short strategy for working with PHP control structures, while similarly remaining ordinary for their PHP accomplices.

Sharp edge empowers you to describe your own particular custom requests using the command procedure. Right when the Blade compiler encounters the custom request, it will call the gave callback the explanation that the order contains.

The going with case makes a @datetime($var) arrange which orchestrates a given $var, which should be an instance of DateTime.

# Database seeding

**Database Seeding:**

Database seeding is to initially seed the database with data. Seeding a database is a process in which an initial or first set of data is inserted into a database after installation.

**Introduction**

Laravel use seed classes for seeding your database with test data. This is a simple method. All seed classes are stored in the database. Seed classes may have any name which user wants, but it should follow some sensible convention, such as UsersSeeder. There is a DatabaseSeeder class by default. From this class, you may use the call method to run other seed classes which allows you to control the order of seeding.

**Writing Seeders**

For generating a seeder, execute the make:seeder Artisan command. All seeders generated will be placed in the database directory.

**php artisan make:seeder UsersTableSeeder**

A seeder class only has one method by default: run and this method is only called when the db:seed Artisan command is executed. Within the run method, you may insert data into your database according to the wish of user. You can use two methods:

1.The query builder to manually insert data

2.Eloquent model factories.

**Using Model Factories**

Of course, manually specifying the attributes is time consuming and difficult. Instead, model factories can be used to conveniently generate large amounts of database records. First, you should know how to define model factory for which you should be clear of documentation. Once you have defined your factories, you may use the factory helper function to insert records into your database.

**Calling Additional Seeders**

Within the DatabaseSeeder class, you can use the call method to execute additional seed classes. Call method can be used to break up your database seeding into multiple files so there is no need to store overwhelmingly large files.

**Running Seeders**

Once you have written your seeder, you may need to regenerate Composer's autoloader using the dump-autoload command:

**composer dump-autoload**

db:seed Artisan command is used to seed your database. By default, the db:seed command runs the DatabaseSeeder class, which is used to call other seed classes. However, you may use the --class option to specify a specific seeder class if you want it to run any class specifically.

**php artisan db:seed**

**php artisan db:seed --class=UsersTableSeeder**

You may also seed your database using the migrate:refresh command. This will re-run your migration. This command very is useful for re-building your database completely:

**php artisan migrate:refresh –seed**

# Automatic pagination

Automatic Pagination:  
• Refers to numbering pages in a document.  
• Refers to dividing a document into pages.   
Configuration  
In other frameworks, pagination is very difficult. Laravel makes it an easy task like a breeze. Laravel can generate intelligent links "range" based on the current page. The generated HTML is compatible with the Bootstrap CSS framework.  
Usage  
There are several ways to paginate items. The simplest is by using   
• the paginate method on the query builder   
• an Eloquent model  
Creating A Paginator Manually  
Sometimes you may wish to create a pagination instance manually, passing it an array of items.  
Paginating an Eloquent Model  
You may also paginate Eloquent models:  
The argument passed to the paginate method or used by the method is the number of items you want to display on each page. Once you have retrieved the results, you may display them on your view, and create the pagination links using the render method.  
This is all that you need to do to create a pagination system. This should be noticed that we did not have to inform the framework of the current page. Laravel will determine this for you automatically.  
You may also access additional pagination information via the following methods:  
Methods:  
• currentPage  
• lastPage  
• perPage  
• hasMorePages  
• url  
• nextPageUrl  
• firstItem  
• lastItem  
• total  
• count  
Simple Pagination  
If you are only showing "Next" and "Previous" links in your pagination view, you have the option of using the simplePaginate method to perform a more efficient query. This is useful for larger datasets when you do not require the display of exact page numbers on your view.  
Appending to Pagination Links  
You can add to the string of pagination links using the appends method.  
This will generate URLs that look something like this:  
http://example.com/some?page=2&sort=votes  
You may also use the fragment method:  
This method call will generate URLs that look something like this:  
http://example.com/some?page=2#foo  
  
Converting to JSON  
• The Paginator class implements the contract and exposes the toJson method. You may also convert a Paginator instance to JSON by returning it from a route. The JSON'd form of the instance will include   
• total  
• current\_page  
• last\_page  
 The data will be available in the JSON array by the data key.