# LARAVEL WITH PHP. (MY PERSONAL NOTES)

* **LARAVEL BLADE:**

This is a php templating engine that makes it easy to embed php in laravel’s html codes. To use this engine, the file should be named as follow [filename.blade.php]. Thus the .blade extension must be in the file name.

**@yield(‘’), @extends(‘’) and @section(‘’)/@stop:**

This is used to refactor codes in laravel to reduce the bulkiness of the code. @yield(‘sectionName’) is used in the common file containing most of the html, to label a part that would be refered to from another file.

@extends(‘common file’s path and initial name’) is used to extend or include the common file in the page of choice.

@section(‘sectionName’)/@stop is used to embed a section that wants to be changed in the common file.

* **LARAVEL RAW SQL QUERY::**

**Insert ::** This is done in the route or controller section using PDO query format. Here a static DB::insert(); class is used. Example below:-

**DB::insert(“insert into post(title, content) values (?, ?)”, [“This is the title”, ”This is the content”]);**

**Read ::** This is done in the route or controller section using PDO query format. Here a static DB::select(); class is used. Example below:-

**$results = DB::select(“select \* from post where id = ?”, [2]);**

**return $results;**

**Update ::** This is done in the route or controller section using PDO query format. Here a static DB::update(); class is used. Example below:-

**$updated = DB::update(“update posts set title = ? where id = ?)”, [“Updated title”, 1]); return $updated;**

**Delete ::** This is done in the route or controller section using PDO query format. Here a static DB::delete(); class is used. Example below:-

**$delete = DB::delete(“delete from posts where id = ?)”, [1]);**

**return $delete;**

* **Eloquent [OBJECT RELATIONAL MAPPER] ::**

This allows us to deal with laravel database with ease.

**[How to create a model** :: **php artisan make:model Post -m] :-**

Here the flag[-m] creates a migration that’s related to the post model. The model created above is assumed to deal with the **posts** table in the data base (that is, the default model table is the model’s name in lowercase ending with an ‘s’).

The newly created model is an extention of the model class.

**MODEL PATH ::/app/.**

**[How to change default table name and primary key] :-**

The newly created model assume the name of the table as mentioned above.. it also assume that every table has and “id” column which is a primary key .. Hence the following code changes the default values..

**Protected $table = ‘RealNameOfTable’;**

**Protected $priamarykey = ‘postsid’;**

**[How to retrieve data from database using ELOQUENT] :-**

use App\Post

Route::get(‘/display’, function(){

$posts = Post::all();

OR

//$posts = Post::find(2);

foreach ($posts as $post){

return $post->title; }

**});**

**[How to retrieve more specific data from db using ELOQUENT]:-**

Route::get(‘/find’, function (){

$posts = Post::where(‘id’, 2)->orderBy(‘id’, ‘desc’)->take(1)->get();

return $posts;

});

**[How to retrieve data using ELOQUENT with exceptions]:-**

**LARAVEL ELOQUENT RELATIONSHIPS::**

This is section deals with how laravel relates several tables in the database. The various relationships includes: One to One, One to Many, Many to Many, Polymorphic etc. The type of relationship is usually indicated using a prebuilt class within a newly created public function located at the User.php model

**ONE TO ONE RELATIONSHIP**

This eloquent relationship literally links table together at the point at which one table has something in common with another

***EXAMPLE ::*** Every post concerning a user can be retrieved from the posts table where the user’s id corresponds with the user\_id column of the posts table.

**INVERSE FUNCTIONALITY**

This functionality allows us to pull the inverse of a particular laravel eloquent relationship say: THIS PULLS FRONM THE USERS TABLE

**ONE TO MANY RELATIONSHIP**

This eloquent relation ship type deals with pulling multiple posts that belongs to a particular user. **See App\User and Route\Web [ONE TO MANY]**

**For an example.**

**NOTE:: ‘RETURN’ keyword in functions only returns one item when used in a loop.**

**MANY TO MANY RELATIONSHIP.**

**This type of laravel relationship requires a pivot table!**

**PIVOT TABLE ::** A pivot table in laravel is known as a lookup/linkup table. This eloquent table serves as an intermediary table between two **one to many** tables. **Thus is a table used to relate two other tables.**

***CONVENTION OF CONSTRUCTING A PIVOT TABLE::***

***[In this case, the parent table is the users table.]***

1. **Create a Role Model and create\_roles\_table migration:**

**Php artisan make:model Role –m** *(This creates a* ***Role*** *model and a* ***create\_roles\_table*** *migration at the same time).*

1. **Create a table that relates the users and roles table:**

**To create this table, the convention used is to**

* **Singularize the two table names you want to join (role and user).**
* **Arrange alphabetically (role before user).**
* **Create a migration (***php artisan make:migration create\_users\_role\_table --create = role\_user***).**
* **To the role migration, add a $table->string(‘name’);**
* **To the role\_user migration, add integer(‘user\_id’); and integer(‘role\_id’);**
* **Create a relationship function :: here, you go to the route that comes last alphabetically and create a public function, indicating that that model or table belongs to the model that comes first alphabetically example ::**

**Public fuction roles (){**

**Return $this-> belongsToMany(‘App\Role’);**

**}**

***THE INTERMEDIATE TABLE ::***

The intermediate table here denotes the pivot table. This table is accessed in the route (web.php) file. Hence this shows the time the pivot table was created at.

***HAS MANY THROUGH RELATIONSHIP ::***

This relationship provides a convenient shortcut for accessing distant relationships via an intermediate relation. In other words, this process relates more than three tables using one [which is not a natural pivot table] as an intermediate table.

***POLYMORPHIC ELOQUENT RELATIONSHIP ::***

The polymorphic relationship allows a model to belong to more than one other model in just a single association. Therefore, this denotes a table that is related to two different tables at the same time.

***EXAMPLE::***

1. Create a Photo model and migration.
2. Add the image path, imageable\_id and imageable\_type in the migration and migrate [here, imageable denotes the owner of that particular image. Whether a post or a user].

***MANY TO MANY POLYMORPHIC ELOQUENT RELATIONSHIP ::***

This when a list of unique records are shared between different tables such as tags. To do this, we create a video model along side it’s migration. Then we create a tag model and it’s migration then a taggable model and it’s migration. [The taggable table relates the video, post and tag together].

***LARAVEL TINKER ::***

The laravel tinker is a very important tool used to interact easily with the database using eloquent. This is a command line tool used to test several eloquent relationships, check objects, do the usual CRUD etc quickly using the command line rather than creating fresh routes.

***HOW TO CREATE DATA USING TINKER ::***

1. Php artisan tinker [ENTER]
2. $post = App\Post::create([‘titlle’=>’The post title’, ‘content’=>’The post content’]); [ENTER]
3. To see this just created post, type the name of the variable {In this case $post}.
4. $post = new App\Post; This is used to create a new object/instance of the Post model i.e saving App\Post into $post variable.
5. To use the method in 4 above ::

* $post->title = “The new title”
* $post->content = “The new post content ”
* $post->save(); [To push to the database].

***HOW TO READ DATA USING TINKER ::***

1. Php artisan tinker [ENTER]
2. $post = App\Post::find(2) ALTERNATIVELY
3. $post = App\Post::where(‘id’, 6)->first(); ALTERNATIVELY
4. $post = App\Post::whereId(6)->first();

***HOW TO UPDATE DATA USING TINKER ::***

* $post = App\Post::find(2)
* $post->title = ‘updated title’
* $post->content = “updated post content”
* $post->save() [This save() method is known as data persisting]

***HOW TO DELETE DATA USING TINKER ::***

1. $post = App\Post::find(2)
2. $post->delete(); [This softdeletes the table row with id number 2]
3. $post = App\Post::onlyTrashed()
4. $post->forceDelete() [This deletes the table row permanently]

***LARAVEL MIDDLEWARE ::***

The laravel middleware deals with it security. This can be done by including the Auth class in the page in question. Here the Auth is an alias for authentication class, which is found at \App\Http\Middleware\Authenticate\. And this alias is declare in kernel.php found under Http folder.

**REGISTERING A MIDDLEWARE:**

After making a new middleware using php artisan make:middleware MiddlewareName, we register this middleware in the kernel.php file as indicated in the login laravel app. You either include it in the routemiddleware or in the web middleware.

**HOW TO USE THIS MIDDLEWARE::**

After registering the middleware, you can either use it via the route or via the controller as a \_\_construct method as seen in the HomeController for a login system.

**NOTE:: To pull out and authenticated user from the database, we use $user = Auth::user;**

**LARAVEL SESSIONS::**

***HOW TO SET SESSIONS ::*** firstly, create a request variable, then create the session like so: $request->session()->put([‘edwin’=> ‘master instructor’]); OR session([‘edwin’=> ‘master instructor’]);

***HOW TO READ SESSIONS :*** $request->session->get(‘edwin’);

**NOTE:: For more check HomeController@index UNDER LOGIN laravel app**

**WE DELETE SESSIONS BY USING THE FORGET METHOD OR THE FLUSH METHOD.**

***HOW TO FLASH DATAS::***

Flashing data is very similar to sessions, the only difference is that flashing only stays for one request. For example when you want to give users response to certain request, like in forms E.G [Your post has been created]. **Check HomeController for syntax.**