1439

Laravel PHP Framework

LARAVEL PHP FRAMEWORK

WADHAH

ADEN| NOVAC

المحتويات

What are the requirements of Laravel

You need three programs (ide + git scm + composer)

How to create a larval app

How to create a Laravel project in git bach

Cd c:xampp/htdocs

```
composer create-project --prefer-dist laravel/laravel Laravel-project 5.2.29
```

how to create a virtual host (domain)

there are two files you want to edit (one related to xampp , anther related to windows)

1) Xampp>apache>conf>extra>vhosts
2) Windows>sys32>drivers>etc>hotst (first open the note pad as an administrator then open this file from inside the note pad)
3) Stop and run xampp

Introduction to the files of Laravel

app folder >> there will be much of the coding in it

> http >> the most important folder , where you are going to find the controllers / middleware (security) / requests (which is used for the authentication) / routes (manages the pages and their url and how you move throw them) /
>
> user.php >> is a model so every thing inside it deals with the database , and if you want to create a model they will be inside the app folder like the user.php

bootstrap folder >> means that bootstrap is downloaded automatically to Laravel

config floder >> one of the main folders >> app.php (one of the main failes , we are going to be including things here to use a plugin < things called a providers) . // database.php (is where you set up your database but we won't do it here in the course , instead we will do it in the .env file in the main dir) // mail.php (to set up a mail and mail server)

database floder >> migrations (it automates how we create tables , so it is easier than phpMyAdmin) // factories (allow us to automate a lot of data entry , so they make data) // seeds (they do the same factories but in anther way)

public folder >> is where we are going to have our css files and java script files

resources folder >> the name says it all , is where we are going to have the assets and the compiled css (scss) // views >> they are the html files of your website // vendor (is for git , and he said that's an advanced topic

vendor folder >> when you include a plugin or package in the project they will be installed in the vendor dir and every thing inside belongs to Laravel

.env file >> database information , mail information (people prefer to but database information here because it is not uploaded to git if upload your project to github meaning that it will be ignored .

Gulpfile.js file >>> you write inside it commands for compiling javascript and css files

Routes

Creating Routes

Routes >> are the url of your pages and you define them in the http folder which is located in the app folder >> you define a route by writing the following

```
Route::get('/', function () {
    return view('welcome');
});
```

how to pass parameters from the url into the page

```
Route::get("/images/{id}" , function ( $id )  {
    return " this is the image that has the id numebr " . $id  ;
});
```

(it is okey to use “ instead of ‘ , but it is not okey to use + instead of . , and it is fine to use the same variable name in more than one route)

And you can also pass multiple parameters

(so the idea is to pass parameters to a views and use them as a variables there like for example the username and posts or any things else .

Naming Routes

In Laravel you can give your routes a nick name so you can use it (if you think the actual url is long)

```
Route::get( "/wadah/esam/haider/hussein/mohsen" , array ( "as"=>"me" ,
function () {
    // those two lines are only for testing
    $url = route('me') ;
    return "your url is "  . $url ;
})  ) ;
```

You can use this command to show all your routes with their names

((php artisan route:list))

_____

Controllers >> are classes that deal with the data coming from the database and going to the views , and also dealing with the data coming from the view and going to the database

Creating controllers

We usually put them in the controllers.php file in the controllers folder that is in the http holder , but you can also create a sub folder for each type of controllers

Namespace >> used to avoid having a collision with the function names .

Use >> used to import all the functions in a specific class , (like import in java)

How to make a controller ((php artisan make:controller [controllerName]))

You can also create a controller with all the resources like (delete , destroy , show , store) which will make it easier for you ,, by using this command ((php artisan make:controller –resource [controllerName]))

Routing a controller

You can route a url into a controller directly by adding a route to routes.php in the following way

Route::get (“[name of the route]” , “[name of the controller]@[name of the function in the controller]”) ;

And that’s an example

```
Route::get( "/posts" , "PostController@index" ) ;
```

Passing parameters to a controller from the url

You can do it the same way as we did it directly , that’s an example of how you can make .

In the routes.php

```
Route::get( "/posts/{id}" , "PostController@index" ) ;
```

In the controller

```
   public function index($id)
    {
        return "this is the post numebr ". $id ;
    }
```

Creating a special (resource) route and connect it to a controller

Resource route is a special route that can work with the special resource controller

You create it like this ((Route::resource([name of the main route] , [name of the controller]) ;)) and thin if you list all the routes you will understand how this works

Example

```
Route::resource( "/images" , "ImagesController" ) ;
```

Assume the controller ImagesController has the method show as follows (from the route:list you can see that the images/{images} will be handled by the method show (don't worry if the variable is $id it will work also but I don't know how , however you can change it to $images and it will work too .

```
   public function show($id)
    {
        return " you are in the show method of the images controller and you
are in the  post number " . $id ;
    }
```

Creating a method in the controller

We can easily create a method in any controller and call it from any route manually like this

```
Public function showPost () {
        Return view ('testview') ;
}
```

Views

Are located in resources > views

You can easily create a view by creating a new file in the views directory with the extension viewName.blade.php (.blade is unique extension for laravel files so it allows you to do more things than the normal php files)

And then you can show it by returning a view from a controller or directly from the route

Return view (‘[The name of the view]’) ;

How to show a view that is put in anther folder (it is put in a folder that is inside the viws folder) > you can just simply write (return view (‘ [the name of the folder]/[the name of the view] ’) ;

Passing parameters to a view

You pass variables to views by two methods

1) By using with and chaining (ugly) > return view(‘testView’)=>with(‘id’ , $id) ;
2) By using the method compact > return view (‘ testView’, compact(‘id‘ , ‘username’ , ‘password’)) ; >> (with out “ ”)

Example of the compact way

```
return view ('user' , compact('username' , 'age' , 'country') ) ;
```

(you pass the names of the variables and Laravel will know that those are the name of the variables and will pass their values also)

How to handle them in the view ? you can handle as any variables by {{$id}} which is much easier way than the vanilla php way

Blade

Blade is a templating engine made by Laravel to make writing html code and php code much easier

Master Layout Setup

The idea is to create a master layout page and changing certain content depending on which page you are in .

1) You create a master layout and normally you name it app.blade.php and put it in a separate folder called layout
2) In the variable section of the layout (the section that will be changing from one page to anther) you put @yield('[name of the section]')
3) In the special page that will use this layout you are going to write @extends('layout/app')
4) Then you are going to fill all or some of the sections that you yielded in the app.blade.php file by using @section([name of the sction]) then write the html code and after you finish you write @endsection

Common mistakes I make > not writing ' ' // mistake writing yield instead of yeild

Passing an array from the controller and show it using the blade engine

You create an array using this way $people = ['wadah' , 'esam' , 'haider' , 'hussien']

You pass it the same way as we passed the $id from the controller by the compact method

You write it like the following way

```
@if (count($people) > 0)
  @foreach($people as $person)
      <div class="title">{{$person}}</div>
  @endforeach
@endif
```

You can also use @include('layout/navbar') to include an html part of code that is in anther file

Migrations

Very important concept in Laravel , it simply makes definition of the tables of the database so you can then simply make them by using a php artisan command and they can be applied to any kind of database including sql or oracle database so that way it can very flexible and if you give your project to anther person he can only perform the php artisan command in order to make the database instead of exporting the database from phpmyamdin and then importing it to his project

.env > is a file where all the environment variables will be stored in and this file won't be shared in git hub if you upload it because it is listed in the gitignore file

Creating migrations

First of all you have to change the info that are in the .env file to the suitable info

You run the command php artisan make:migration [name of the migration] –create="[name of the table]"

Then you open the file from the migration folder that is in the database folder and start adding the columns of the table according to the convention ,, >> you can use help of the Laravel documentation https://laravel.com/docs/5.0/schema

Until now the migrations are not in the database yet , you should perform the command in order to perform it php artisan migrate

Note : the tables that are already migrated will not migrate again when you perform php artisan migrate

If you want to remove everything you did in the last migration , you can do it by using the command php artisan miagrate:rollback

Important if you want to change a table (add a new column to it for example) , you can do that by creating a new migration by the command (php artisan make:migration [name of the migration] –taple="[name of the table that you want to edit]") then you go to that file in the migration folder and but $table->string('blah') just like you put columns in the migration and then migrate the project and the result will be shown

you can reset the whole database to its initial state by running the command (php artisan migrate:reset)

Queries with Laravel

There are many ways to deal with inserting or deleting or updating data inside the database one of them is the normal sql queries which is the hardest one , but you can do any thing and every thing with id

Insert queries

In order to insert something to the database you have to run this line of code (it can be inside a routes.php and you can't put it in the controller for some reason)

DB::insert(' insert into posts (content , title) values (? , ?) ' , ['laravel' , 'laravel is the best thing that happened to php '])

The best thing when you do them in the Laravel way that they are very secure and protected .

Reading Queries

You define a variable that will catch the result from the select method that is going to return the results of the query

The select method is written as follows DB::select('select*from posts') ;

Then you deal with the array that is returned as you deal with any other array

That's a full example of the select (it's not going to work as expected because the return will stop the function when it is encountered first time)

```
Route::get('/showPosts' , function () {
    $posts = DB::select('select * from posts') ;

    foreach($posts as $post) {
        return $post->title ;
    }

});
```

Updating queries

As expected the updating method is the nothing unusual , just you can keep in mind that the update method will return the number of rows that are affected

Example

```
Route::get('/updatePost', function () {
    $numOfRowsAffected = DB::update('update posts set title = "java" where
title = ? ' , ['php'] ) ;

    return $numOfRowsAffected ;
```

```
});
```

Deleting Queries

Is the same exact thing as the updating

```
Route::get('/deleteFirstPost' , function () {
    $numOfRowsDeleted =  DB::delete('delete from posts where id = ?' , ['1']
) ;

    return $numOfRowsDeleted ;
});
```

Eloquents

Create Models

You should first create a model by running the command php artisan make:model Post this command creates only the model with out creating the migration of that table , so if you want to create the table with its migration you run the command php artisan make:model Post -m

You will find the models' files in the app directory . like the user model

Note : if you create a model with a name (Image) it will assume that you have a table with the name (images) > (all small letters) , and if you create the model with migration at the same time , it will also name the table as (images) . also the model assumes that the primary key of the table is the column (id) but if you want to change it later with the name of the table you can do it like the following .

```
class Post extends Model
{
    protected $table = "posts" ;
    protected $primaryKey = "id" ;
}
```

But normally you should keep them in the same way as this , so you don't miss the convention

How to Use Models

You can use models in the routes or in the controllers , first of all you should import the model in the file you are going to use it in like the following way

```
use App\Post ;
```

then you can use the model methods as you want like this way

$posts = Post::all() >> this will return all the records that are in the table post and put it in the variable posts then you can take them from there as we already saw in the sql queries .

You can use anther method which is find([the id you want to retrieve]) this method will retrieve the post that has the id as the parameter

A full example :

```
Route::get('/find/{id}' , function ($id) {
    $post = Post::find($id) ;
    return $post->title ;
} ) ;
```

Note The letter (A) in app should be capital ,

Conditional Reading with models

This is an example on how you can write a full query in the Laravel way , it is easier but not fully customized and needs a lot of practice

```
$posts = Post::where('title' , $title )->orderBy('id' , 'desc')->take(2)->get() ;
```

Very Important >> you have to write the get() method at the end of the code so you can catch them in the variable .

Otherwise the other methods are clear to understand

There are many other methods that you can use in eloquent

For example you can use where like the following way where('id' , '<' , 10)

Or instead of get() you can use first() , so you can only retrieve the first element

Inserting and updating with Eeloquents

You can save a new eloquent by writing the following in the controller or route closure function but note that the model has to be declared up in the code (use App\Post) then you can write the following to save a new record :

1) First create a new object of the mode ($post = new Post) >> not like the java $post = new Post () ;
2) Then start filling the columns of the record in this way ($post->title = 'this is a title')
3) Then save the record by applying the method save of the class model to the object ($post->save())

So the whole code when you look at it , will look like the following :

```
public function insertPostEloquent ($title , $body) {
    $post = new Post ;
    $post->title = $title ;
    $post->body = $body ;

    $post->save() ;
}
```

If you want to use the same way for updating any field you should return it first and retrieve it by a variable and then modify it the same way , then save it , and it will be updated . Example :

```
public function changeTitle ($oldTitle , $newTitle) {
    $post = Post::where('title' , $oldTitle )->first() ;
```

```
        $post->title = $newTitle ;
        $post->save() ;
    }
```

Creating Data

For some reason that I don't really know currently , you can create data and put them in the database the same as we did in the inserting stage , but Laravel as Edwain said doesn't allow create to work out of the box , so you should allow it to work from the model it self .

He said that create works with the mass assignment (I might know more about it in the next chapters)

So in order to enable create to work for a specific model , you go to that model and you change the property fillable (which is an array) in this way (protected $fillable = ['title' , 'body'] ;) << here you write the columns that you are going to be using with the create method . if you don't do it you will get a mass assignment exception when you use the create function or you can define the $guarded array instead of fillable but you should define the attributes that shouldn't be mass assigned

Know you can use create like this >> Post::create(['title'=>'this is a title' , 'body'=>'this is a body']) ; then the record will be created and will be put in the database directly

Note that you do this one (=>) for the arrays , and you do this one (->) for the methods .

Updating with eloquent

We took a way how to update one record and save it , and this way is not bad and with a little tweaking you can make it work for more than one record , but there is anther way to do it , which is with the update method , you first get all the records that you want to update and then instead of using the get() method you use the update() method . like the following (note that the update function takes an array) : (Post::where('title','php')->update(['title'=>'this is edited title' , 'body'=>'this is edited body']) ;)

Deleting Eloquents

Is very easy and can be predicted if you understood the inserting and the updating ,, and as the updating it can be done in two ways

Those are the three ways and you can choose any one of them to do it < the third way is used only if you have the id of the record it can be used to delete one record also .

```
    public function deleteFirstPost () {
        // this way is working perfectly
        // Post::all()->first()->delete() ;
```

```
        // the second way
        // $post = Post::all()->first() ;
        // $post->delete() ;

        // the third way
        Post::destroy([8,9]) ;

        return 'first post deleted' ;
    }
```

Soft deleting and trashing with Eloquent

Soft deleting is a very cool concept , where you can delete an item but it is still in the database so you might want it back in anther time . so the idea is to enable soft deleting for a specific model because it is disabled , and you tell the model that this column is actually a date , then you add the column throw migration and migrating the migration :

So :

1) Enable soft deleting by writing above with the other uses (use Illuminate\Database\Eloquent\SoftDeletes;) and this inside the class (use SoftDeletes ;) in the model that you want it to use it . with the s
2) Make a migration to add (deleted_at) column and run it , you do it like the following ($table=>softDeletes()) ; with the s
3) Let the model know that this column is actually a date (protected $dates = [‘deleted_at’] ;) with the s
4) Delete the records the normal way you delete them , then if you go to the database you will still find them but they will have a date inside the deleted_at column instead of the null value .

I think It is important to do the steps in the same order

Note for some reason I can’t delete the same way I was used to delete data , I have to use the find function , like this > Post::find(1)->delete(); it won’t work if you use it that way Post::all()->first->delete()

The method all doesn’t work with the soft deleted items and it is kind of understandable

Retrieving deleted records

The whold idea is to use (withTrashed) and (onlyTrashed) methods

For example : Post::withTrashed()->where(‘title’,’php’)->get()

Restoring deleted records

Restoring a safe deleted items is done via the method restore() . you first have to get the row that is already delete it and then chain the restore method to it

```
    public function restoreSoftDeletedPost ($id) {
        $posts = Post::onlyTrashed()->where('id',$id)->get() ;
        $posts[0]->restore () ;
        return 'the post with the id '.$id.' is restored ' ;
    }
```

Deleting a record permanently

If you are sure that you don't need the record that was actually been safe deleted , you can delete it forever via the method forceDelete() like the following way

```
Post::onlyTrashed()->where('id',$id)->first()->forceDelete() ;
```

Eloquent Relationships

One to one Relationship

You can relate two tables (models) in a one to one relationship so for example user has one picture or something like this , in a very easy way so you don't get stuck writing a lot of queries .

So the idea now is to let every user has one post (make a one to one relationship with the user and the post tables) , Steps :

1) First you need to add the column user_id to the post table using the migration , in this way , $table->integer('user_id')->unsigned() ;
2) Then you need to add a function to the user model (the parent) and tell it that the user must have a one post . function post () { return $this->hasOne('App\Post') }
3) Then you can pull the post of any user using this function , for example if you want to pull the post of the user that has the id number 1 you can do it like this return User::find(1)->post;

Notes : 1) you have to make sure that you did the use App/User where ever you make contact with the user database 2) you have to name the column with the name (user_id) because it is the default name >> you can name it with anther name but you have to change the method that is in the user model file . 3) very Important focus that you should use the forward slash not the back slash in App\Post

The Inverse Relationship

Is actually the inverse functionality of one to one relationship so you can get the user of any given post , and it is simple and predictable . so all what you have to do is you go the post model and defile a function user () and put inside it the following code return $this->belongsTo('App\User') ;

And then the exact same thing with one to one relationship when you want to pull the user of the post out of the database .

One to many relationship

It should be super easy and clear by now that we are going to repeat what we already did previously but instead of the method hasOne() we will have the method hasMany() and we will retrieve the posts as an object of objects

Many to many relationship

Basically the same concept but instead you will have to create an intermediate table that has a name derived from both tables that you desire to make a relation between them . and then you use the method belongToMany()

Important that you name the intermediate table with the singular form of the two tables and ordered in an alphabetical order .

You can pull data out easily like other relationships

```
$users = Rol::find($id)->users ;
```

Querying the intermediate table (pivot)

Every thing is really obvious in the documentation and I don't think I am ever going to use it but it is cool to know it thougt

As you have already learned, working with many-to-many relations requires the presence of an intermediate table. Eloquent provides some very helpful ways of interacting with this table. For example, let's assume our `User` object has many `Role` objects that it is related to. After accessing this relationship, we may access the intermediate table using the `pivot` attribute on the models:

```
$user = App\User::find(1);

foreach ($user->roles as $role) {

    echo $role->pivot->created_at;

}
```

Notice that each `Role` model we retrieve is automatically assigned a `pivot` attribute. This attribute contains a model representing the intermediate table, and may be used like any other Eloquent model.

By default, only the model keys will be present on the `pivot` object. If your pivot table contains extra attributes, you must specify them when defining the relationship:

```
return $this->belongsToMany('App\Role')->withPivot('column1', 'column2');
```

If you want your pivot table to have automatically maintained `created_at` and `updated_at`timestamps, use the `withTimestamps` method on the relationship definition:

```
return $this->belongsToMany('App\Role')->withTimestamps();
```

Inserting data into the pivot table

The idea here is to attach the roles into a specific user or the opposite , so the user should first exist in the data base , then you can do the following to attach some roles to him

```
$user = User::find(1) ;
$user->rols()->attach(['2' , '1']) ;
```

Has many through relationship

If your app is like the following ,, every user has posts and every user has also a country , then you can get all the post for a specific country .

You create the country table and but the country_id in the users table , then in the country model you create the function posts (so you can get all the posts that has been written by people of this country) inside the function post you write the following

Return this->hasManyThrought('App\Post' , 'App\User') ;

So the first parameter is what you want to get , and the second parameter is the intermediate table .

Then when you want to pull this information out you write the following

Polymorphic Relationship

Imagen you have a websites that shows videos / photos / posts each in a separate place , and you provide the users the functionality to comment on them , so you will have to put a comments table for each one of those types because you will have post with id number 1 and a picture with id number 1 , so it can be harder to detect which one do you refer to . Laravel provided us with the polymorphic relation ship to solve this dilemma , so you can have one comments table that covers all three types .

1) First you will need to create the comments model and migration but don't you will have to add the columns (commentable_id , commentable_type) , the commentable_id would be the id of the commentable (that could be either photos or posts or videos) and the commentable_type is the model path so if the commentable is a post then it is App\Post .
2) Now you will have to change the all the including models each according to its role in the relation
3) Go to the comment table and add a method called (commentable()) and inside it will be like this (return $this->morphTo() ;) ,, that's the only thing you will have to do .
4) Go to the post and user models and add to them the function comments() and write inside it (return $this->morphMany('App\Comment','commentable') ;) >> the commentable here refers to the function that you made in the comment model .

Then if you want to get the comments of a post or the comments of a video you can do it simply the same exact way as we have been doing it all the time .

It is easy also to retrieve the owner of a specific photo from the photo it self like the following $photo->imagable() but you should be careful on how you deal with the object that is returned because it could be either a post or a user , so be careful

Many to many polymorphic relationship

Imagen you have tags that can be given to posts , photos , (any other thing) so each photo can have many tags , and each tag can be given to many photos .

The idea here is to create a taggables table that has three columns tag_id , taggable_id , taggable_type .

Then In the tag model you create a function for each taggable type for example one function for the (photos) then you will name it (photos()) and you will put in it the following (return $this->morphedByMany('App\Photo' , 'taggable') ;)

And the same as before , in each of the posts and the users you can create a methods tags that has the following (return $this->morthedByMany('App\Tag' , 'Taggable'))

It might seem a little bit confusing but if you go to the documentation you can follow it and will be so easy , and normally the many to many polymorphic relationships are used for the tags

Tinker

It is a command tool that works with php artisan , helps with creating data in the data base , and testing stuff

You go the command and type php artisan tinker and it will open a space for you that you can test every thing out , and it you write it the same way you do it with the normal manipulation of models .

How did I removed the public from the url

1) Inside the server.php >> remove the public from there
2) Move .htaaccess and index.php from public to the main folder
3) replace two lines in index.php as

```
require __DIR__.'/bootstrap/autoload.php';
$app = require_once __DIR__.'/bootstrap/app.php';
```

Forms and validation

Create a form normally like any other form and put the method=POST .

Then you can catch the post request that will come to you as a Request object $request . you can do a $request->all() or $request->title (title is the name of the input so be careful with your names)

Fixing the token mismatch > I added this code to the form code and it worked for me

```
<input type="hidden" name="_token" value="{{ csrf_token() }}">
Or instead you can write
{{csrf_filed()}}
```

After the controller finishes its job it can redirect you to anther page by using the method redirect() >> and this method should be written after the return statement

Return redirect('posts') ;

If you want to have a better representation of your data after you send them to the view you can put them in a dd method dd()

You can route to any place from the html blade files by doing the following (route('posts.edit , $post->id) ;) , this will direct and also attach the id >> this method is magical but you should have a name for the route so you can get to it .

You can use the edit page to show a single post and provide an input to let the user change the content and then after this is done the form should send the data to the posts\update via a put method ((in order to tell Laravel that the method is put you have to put a hidden input tag with the name="_method" and the value="PUT")) and also don't forget the csrf token

Solution for phpMyAdmin 1544 error

the error title : Uncaught Error: Call to a member function exists() on null

you have to open the phpMyAdmin page where the error is shown then press f12 then go to application then clear storage

Laravel collective

Laravel collective is a package so you first should install it from the internet , so in order to do that you should do the following

1) go to the package website to see which lines you should add to the project .
2) modify the file composer.json that is in the main directory.
3) Modiy the file app.php to add providers and aliases
4) Run the command composer update

This is a good package for writing forms , it makes it easier specially the form's header since it creates the csrf and if your method is put or delete it does the hidden input for you

And it has a something great , which instead of writing the url where the form should submit , you can do something else which is submit the form to a specific method in a specific controller directly , and the method will detect which route you meant. Also it provides good linking

But for me the other functions of this package is useless sense the input components are easy to remember and write

Validating forms

In the method of the controller that handles the $request you can write this code >>

```
        $this->validate($request , [
            'title'=>'Required|Unique:posts|Max:10' ,
            'body'=>'Required' ,
            'user_id'=>'Required|Numeric|Exists:users,id'
        ]);
```

If an error happened there will not be anything shown in the page (the page will still be as it is) but all the errors will be saved in an error variable $errors

Creating your own request

You can create your own request with its validations so that you can use it in many places (for example In my case I had to write the validation for creating and updating the post request twice but now with using this you can create it only one time and then reuse it)

You create a new request by this command (php artisan make:request PostRequest)

1) Then you make the authorize method return true .

```
    public function authorize()
    {
        return true;
    }
```

2) Then you create your own validation rules in the rules method

```
    public function rules()
    {
        return [
            'title'=>'required|unique:posts,id|max:10' ,
            'body'=>'required|max:200' ,
            'user_id'=>'required|exists:users,id'
        ];
    }
```

3) Then you use that class in the controller that you want to use it in

```
use App\Http\Requests\PostRequest;
```

Dealing with dates using carbon

You have first to use the carbon class

Use Carbon\Carbon ;

Then you can use the carbon methods

Here is the documentation on how to use it

https://carbon.nesbot.com/docs/

More About Database and Eloquent

Accessors

Some times you want to do something to the data before showing it to the user , and you might repeat this thing over and over again > why repeat it while you can use accessors , what are they ? , (imagen you have a data base of users in your database , and you when ever you pull a name from the database you want it to have a first letter capitalized) , all what you have to do is to make an accessor which is a function in the user model . this function should have a name which follows the convention > get[name of the column with capital first letter]Attribute($value) > for example getNameAttribute($value) inside it will be something like this return strtoupper($value)

; to change all letters to upper case letters , and there are a lot of methods to do similar things , and the value variable is the data that will be retrieved from the database.

Mutators

Mutators allow you to modify data before saving it in the database (or updating) , so it is the opposite of the accessors , and creating a mutator for an attribute is the same as creating an accessor but instead of writing a get you write a set in the mutator function , but inside the function is a little bit different

Query Scopes

Think about them as a shortcut for eloquent methods that you create

So instead of you writing every time Post::where('id','>','10')->orderBy('id','desc')->get() > you can only write one method , lets find out how to declare this method .

You go to the model you want to create the query scope in , then you add a static function with this convention public static function scopeLatest($query) { return $query-> where('id','>','10')->orderBy('id','desc')->get() } so if you want to call this method you will have to do it directly in any controller>> Post::Latest()

Fix for Laravel not compatiable with php 7.2

count(): Parameter must be an array

In order to solve this you must put this code before your code in the controller . or put it in the beginning of routes.php or web.php >> but with this way you are eliminating all the warnings so you won't have them in the future .

```
if (version_compare(PHP_VERSION, '7.2.0', '>=')) {
    // Ignores notices and reports all other kinds... and warnings
    error_reporting(E_ALL ^ E_NOTICE ^ E_WARNING);
    // error_reporting(E_ALL ^ E_WARNING); // Maybe this is enough
}
```

Dealing with files in the forms

Setting up the form

You can let your forms send files to the backend by setting the 'files'=>true and putting the files input with the other inputs and give it a name .

Retrieving the files in the backend

How to deal with the file in the database? You can catch it by $file = $request->file('[name of the file']) ; > then there are many methods to deal with the file . for example getClientOriginalName() to get the name of the file > and if you return the $file object by it self you will get the temporary link of this file > and if you want to get the size of the file you would use the method getClientSize() .

Storing the files

You can store the files in the files easily by calling the method $photo = move('[name of the folder']) > and this method will return the new path of the image so you should save it if you want to use it later or save it in the database

```
        $post = new Post ;
        $post->title = $request->title ;
        $post->body = $request->body ;
        $post->user_id = $request->user_id ;
        if ( $image = $request->file('image') ){
            $path = $image->move('images') ;
            $post->image = $path ;
        }
        $post->save() ;
        return redirect('/posts') ;
```

you can check wither the file is an image or not > also you can check that there is a file before you but it in the database

Authentication

Creating authentication

In order to add authentication to your project you should have a fresh Laravel installation , then you should migrate the existing users and reset password tables , then finally you have to run the command (php artisan make:auth) .

When you run this command you wll get many views related to the authentication and then you can just control stuff

You can access the data of the user at any time by the class auth like the following > Auth::user()->name you can pull other attributes the same way as well > you don't even have to send the user's data from the controller .

The method Auth::check() will return true if the user is logged in and false if it is a guest , and you can use the other methods too

Middleware

Most of the redirecting is done in the middle ware , it is a way of protecting data from unauthorized users > they can be underaged or doesn't have the admin role or any other things .

You can create a middle ware by entering the command (php artisan make:middleware [name of the middleware) , if you want to give it a special name (alias) like the (auth) middleware , you can do that in the kernel.php file ,

Then you have to declare your middleware in the kernel.php in order to use its short name , then in the handle method of the middleware you write what ever you want to be handled (this method will be called whenever you try to access a route protected by this middleware

This line return $next($request); lets the application move farther which means that if the user fulfils the conditions you specified , this code will run ,

You can run your conditions either before or after you move the user to his next place

How to register a route with a middleware (there are many way , either by putting an array or by chaining)

```
Route::get('/', function () {


})->middleware(['first', 'second']);
```

Putting your website in the maintenance mode

You can do the following command > php artisan down

Sessions

```
Route::get('home', function () {

    // Retrieve a piece of data from the session...
```

```
    $value = session('key');

    // Specifying a default value...

    $value = session('key', 'default');

    // Store a piece of data in the session...

    session(['key' => 'value']);

});
```

If you want to check if the user has specific session in his sessions

```
if (session()->has('users')) {}
```

if you want to delete a specific session you can do it as the following

```
session()->forget('key');
```