LARAVEL INSTALLATION AND PROJECT CREATION

****

**** THIS FOLDER MUST BE INSIDE XAMPP/HTDOCS *******

****

concept of namespace in normal php file.

1. with same file name and same class inside file, then we have to change the name of a file among those

2. else you can follow these:

a. use namespace Abc; //abc can be replaced with any other name as namespace;

under namespace Abc we are creating a function or everything. Creating namespace same as foldername.

b. using same function and class file copy paste same file into another Pqr folder.

then namespace Pqr; //Pqr is the foldername, and we can't use one.php as require Abc/one.php as because it has to be used with it's namespace.

we can do as: $t=new Abc\one(); // Abc here is the namespace and one() is function

c. we also can use namespace OR import the file with with;

use Abc; //here Abc is namespace.

use Abc\one; //here Abc namespace with class name one.

ALSO CAN BE CREATED OBJECT AS:

$t1=new Abc\one();

$t2=new Pqr\one();

UPLOAD A PROJECT IN GIT WITH COMMAND LINE:

1. MOVE TO YOUR PROJECT FOLDER IN CMD

2. --> git init

3. --> git add .

4. --> git commit -m "first commit" // instead of first commit anything can be written, double quotation is necessary

```
if(errors after 4th step such as global user & email){
        then use step 5 and 6
} else{
        skip 5 and 6;
        goto 7;
}
```

5. -->git commit config --global user.email 071.sabin.panthi@gmail.com

6. --> git commit --global user.name 071Sabin

====> 5 and 6 steps are just for the first time, after it's done no need for second time.

7. --> step 4

8. --> git remote add origin <github-repos-link>

9. --> git push <github-repos-link>

10. --> have to login for the first time.

11. --> done!!

After this is done for the first time: something may be edited in local project.

So for that just write:

Cmd→git add . (this adds all the files to your git on local machine)

Cmd→git commit -m “anything here”

Cmd→git push

That’s it

BOOTSTRAP

bootstrapmade.com

bootstrap.min.js --. this file has code with eliminated spaces. so min.

1. bootstrap, rows and columns for align items 3,4,5 in a row. 12 can fit in a row.

*js has to be loaded just before the body closes.

2. favicon icons??

3. https://getbootstrap.com/docs/5.3/getting-started/introduction/

4. md--> medium devices, col-md-6

5. my ---> margin top and bottom

6. mx --> margin left and right.

7. bootstrap gives browser compatibility

8. <meta> tags are for SEO optimization

Download php and then install composer.

WHEN YOUR PHP FILE DOESN'T RUN, CHECK YOUR PHP VERSION AND CHECK VCC VERSION.

PARTICULAR WINDOWS MAY HAVE PARTICULAR VERSION OF VCC THAT MAY RUN CERTAIN VERSION OF PHP BUT NOT THE HIGHER VERSION.

We may find newer php version not running in our device in future.

SETTING UP FOR LARAVEL INSTALLATION:

1. Download and install **php**, set **environment path** direct to the php folder; path\phpFolder
2. Download and install **composer**, this is what we need to install laravel.

XAMPP/PHP/php-ini *file must be enabled these things: Or* ***php.ini*** *your different, php installed folder.*

//we also can edit the execution times and file uploading sizes etc.

max_execution_time=120

memory_limit=512M

post_max_size=100M

//these must be there(remove semicolon before these, remove comment)

extension=bz2

extension=curl

extension=fileinfo

extension=gettext

extension=mbstring

extension=exif

extension=mysqli

extension=openssl

extension=pdo_mysql

------> *check if these changes are working or not:- http://localhost/dashboard/phpinfo.php*

INSTALLING LARAVEL AND PROJECT CREATION:

1. INSTALL PHP AND COMPOSER FIRST, DEFAULT IT WILL BE INSTALLED IN PHP FOLDER DEFAULT.

2. USING THESE COMMANDS, WE CREATE A PROJECT.

    //installs laravel globally, it's just first time installation in your pc.

    cmd --> **composer global require larvel/installer**

    cmd --> **laravel new <ProjectfolderName>** *//creates project folder.*

3. some errors may arise while executing the first command above. Such as enabling something in php.ini, so go to php installed(not of xampp) because **php.ini** file is edited of the downloaded php, not of Xampp php.

    and then enable those extensions showm in error while installing laravel or creating project.

4. and then go to the project directory just created using command(laravel new <prjname>)

5. after navigating to prj.folder, cmd --> **php artisan serve**

6. SERVER STARTED.......

HOW TO RUN THIS PROJECT?

while using xampp we used localhost:3000..... OR using directly php file as url= **localhost/foldernavigation, but in php:**

1. we use **'artisan'** as to run server in php-laravel.

2.cmd → **php artisan serve(**open the project folder in vsc and then open cmd there.**)**

SERVER STARTED….

3. After starting the server in cmd, we can open the project folder in vsc and open a new cmd in vsc.

4. open a browser and url= **localhost:8000**

5. laravel simple home will display(or the given path on web.php will display).

6. except public folder everything is secured, all our coding are secured.

FINDING MVC IN YOUR PROJECT FILE;

app/http/controller ===> here will be out controllers, we will inherit from this controller.php.

app/models ===> we will create our models here.

resources/views/anyfile.php ===> this will be views, anyfile inside views folder.

*****************these are the three locations where we practice MVC.***

config ========> settings related to DB, file systems, sessions etc.

config/app.php =====> general settings of our project is inside this file.

WRITE CTRL+T IN VSC AND TYPE FILE NAME TO QUICK ACCESS.

WHAT'S INSIDE →CONFIG/APP.PHP?

Inside .env file;

1. env('APP_NAME', 'abc') ====> it means goto .env file and extract APP_NAME if can't extract, use abc.

INSIDE APP.PHP

2. **APP_KEY** ====> random key used for encryption. Everyone has different keys.

3. **APP_DEBUG** = true; //if false it shows 404 or sth else error.

4. **APP_URL** ===> host name.

|**********change timezone in .env file to Asia/Kolkata.***

|**********change APP_NAME=Video_Site in .env file(can't be space in btn video Site).***

mysite.com/register.php

mysite.com/login.php

mysite.com/register

MOVE TO ROUTES FOLDER:

THIS HELPS TO NAVIGATE TOWARDS THE FILES. YOU CREATE **ABC.BLADE.PHP** FILE.

THEN ROUTE TO THE SAME **abc.blade.php** file as:

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

****** controller.php controls everything.***

php artisan make:Controller HomeController *// this will create a home controller in the controller.php folder.*

After creating home controller, use/app..... file path.

also create **UserController.php**

create one one functions here in both controller and **usercontroller.php**

It's explained in details below………>>

MANAGING HTML FILES AND IT'S NAVIGATIONS:

1. Inside **resources/views** in your project folder, you have to create a new file as **filename.blade.php**
2. Write any html code in it, multiple codes can be created.
3. You can create as many files as you want.
4. Suppose you created abc.blade.php file in **resources/views,** THEN ROUTE TO THE SAME **abc.blade.php** file as:
   ```
   Route::get('/', function () {
      return view('abc');
   });
   ```

5. Inside the **routes/web.php**, This code below routes to
   ```
   Route::get('/', function () {
      return view('welcome');
   });
   ```

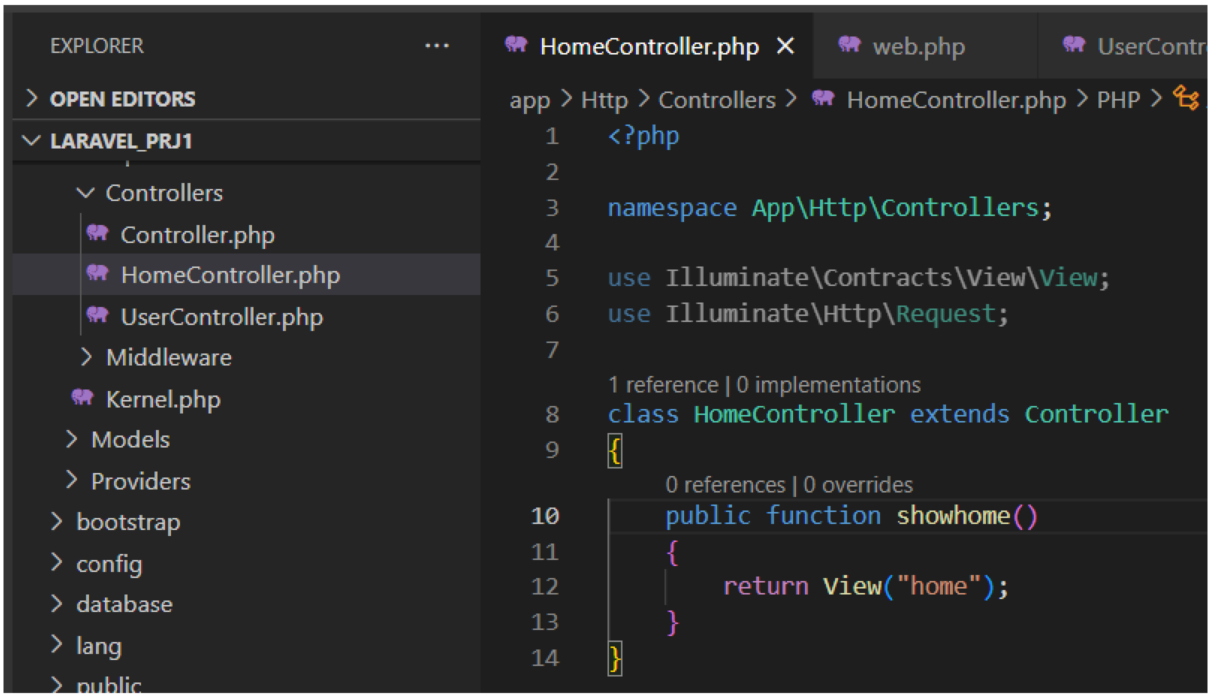
Code above explains that it wil be navigated to default **welcome.blade.php** file on **localhost:8000/** loading in url.

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

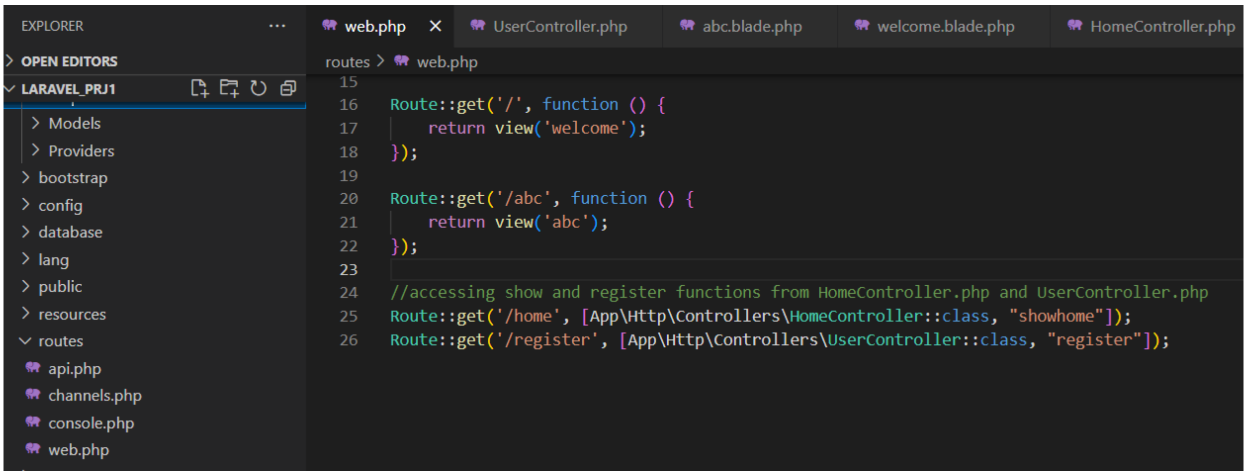
To run above code, you must create a **register.blade.php** file inside **resources/views**.
Like this you can create as many as .blade.php files and link to the **web.php.**

CREATING A NEW CONTROLLERS AND NAVIGATING TO IT'S FUNCTION:

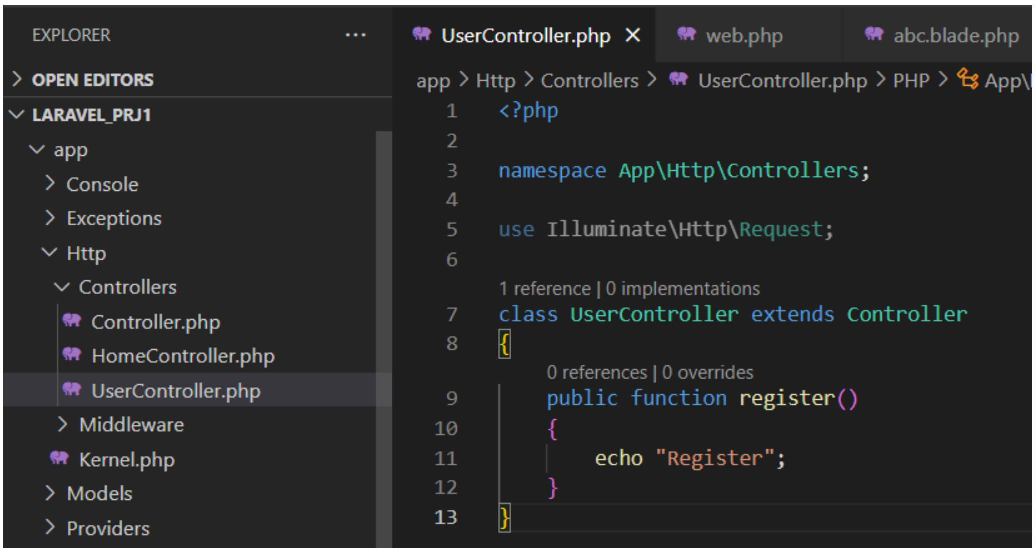
1. Create a new controller from command prompt, as **Controller.php** is the default php file. We are going to create our own controller as:
   php artisan make:Controller HomeController *// this will create a home controller in the controller.php folder. There create your own method and this controller has to be imported using namespace into* ***routes/web.php file****. Homecontroller file extends controller, use it's features with our new HomeControllers. So that no need to edit the default controller.*
   THESE CONTROLLERS ARE CREATED INSIDE ***app/http/controllers.*** *This path is mentioned in MVC.*
   ****we are just using route names to route because using names we can just change the path in* ***web.php*** *only. Else we have to go to respective file and then change path everywhere.*
2. Create one more **UserController.php** file and create one more method there.
3. Inside **web.php** file route to the same **HomeController.php** and **UserController.php** file as:
   `Route::get('/', [App\Http\Controllers\HomeController::class, "showhome"]);`
   Here, **showhome** in above line is the function inside **HomeController.php**.
   Like this many files can be routed. FIG BELOW for showhome function.



```
<?php

namespace App\Http\Controllers;

use Illuminate\Contracts\View\View;
use Illuminate\Http\Request;

1 reference | 0 implementations
class HomeController extends Controller
{
    0 references | 0 overrides
    public function showhome()
    {
        return View("home");
    }
}
```



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

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

//accessing show and register functions from HomeController.php and UserController.php
Route::get('/home', [App\Http\Controllers\HomeController::class, "showhome"]);
Route::get('/register', [App\Http\Controllers\UserController::class, "register"]);
```



```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class UserController extends Controller
{
    public function register()
    {
        echo "Register";
    }
}
```

This **UserController** file has a **register** function and it is navigated in **web.php** as:
Route::get('/register', [App\Http\Controllers\UserController::class, "register"]);

DAY 3

Vendor folder can be deleted while sharing the project.

- ➔ **Composer.json**=list of libraries that composer has imported. *Vendor folder can be deleted while we share our projects with friends.*
- ➔ **Composer install** can be run in cmd after sharing the file with friends and **vendor** folder is again created.
- ➔ **Composer update** command is also there, it will look at the list and instead of downloading that library, it will download the latet verison of that library.
- ➔ Creating **file** inside controller folder as ➔ cmd—**php artisan make:controller user\controllername**
- ➔ We have to import every classes in whatevercontroller.php

CREATED A NEW PROJECT:

Route::get('/user/login', [UserController::class, "login"]);

Click on **userController** and press ctrl+alt+I to import the class **usercontroller.php** automaticaly.

(it's done inside web.php file) pictures are given below:::---

*debugging function dd==> dump and die.

*say to print just x as: **{{ x }}**

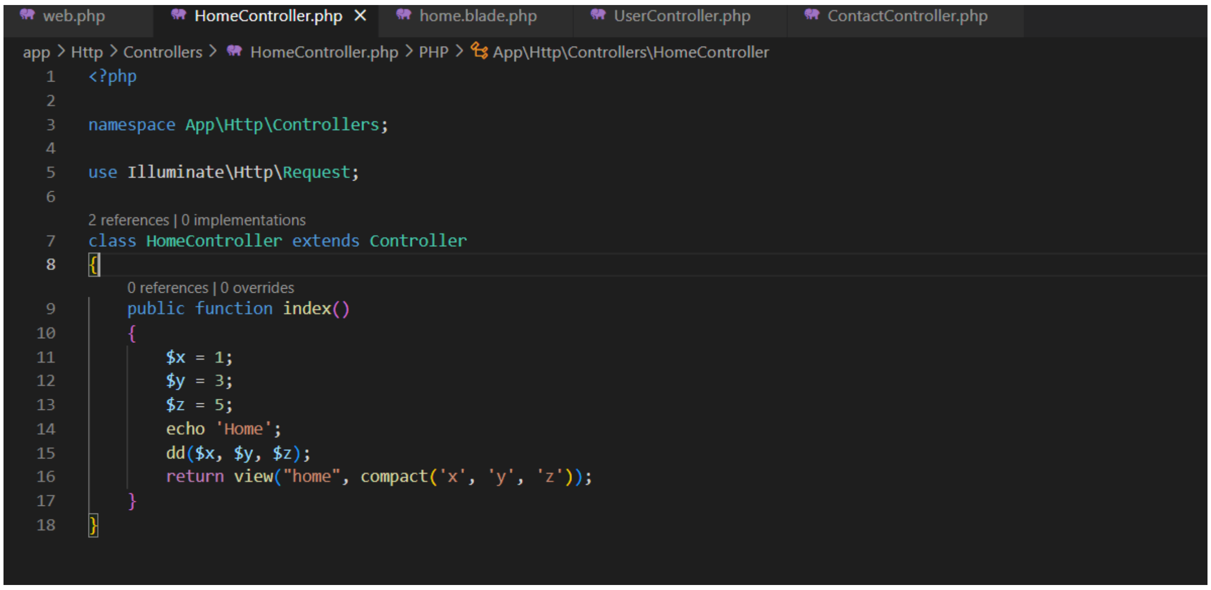
*.blade.php file filters to stop execution of js and html files inside a string declaration.

As: $s="sabin <anyJSCode>";

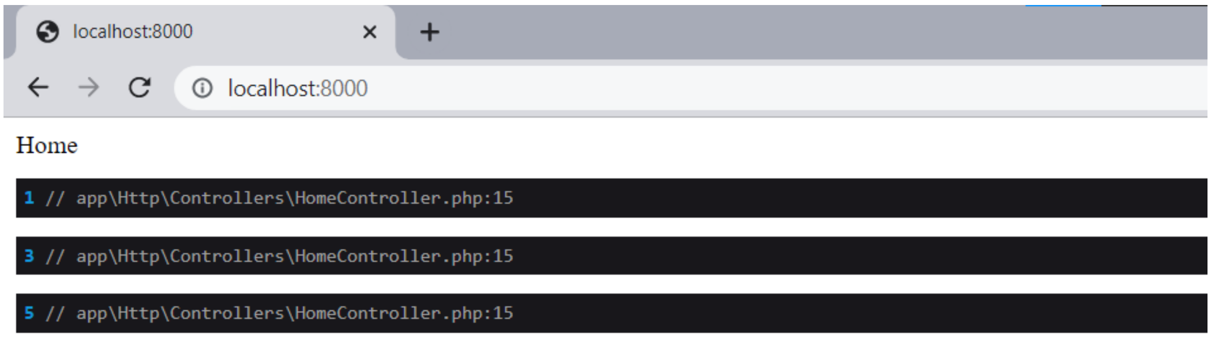
So blade file discards <anyJSCode>-----if any js code is written there then it would execute it, but blade file prevents it.

*inside blade file writing if else condition as:

@if($1==1)


web.php
HomeController.php
home.blade.php
UserController.php
ContactController.php
app > Http > Controllers > HomeController.php > PHP > App\Http\Controllers\HomeController

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

2 references | 0 implementations
class HomeController extends Controller
{
    0 references | 0 overrides
    public function index()
    {
        $x = 1;
        $y = 3;
        $z = 5;
        echo 'Home';
        dd($x, $y, $z);
        return view("home", compact('x', 'y', 'z'));
    }
}
```



```
Home
1 // app\Http\Controllers\HomeController.php:15
3 // app\Http\Controllers\HomeController.php:15
5 // app\Http\Controllers\HomeController.php:15
```

YOU CAN SEE EVERYTHING IN LARAVEL WEBSITEWHATEVER SYNTAX ARE GIVEN BELOW

Thse are written inside blade.php file.

@if($x==1)

    Hey 1

@elseif(aaa)

    Hello aaa

FOR LOOPS:

@for($i=1; $i<10; $i++)

*copy the index.html codes from bootstrap and paste in home.blade.php.

*paste your assets folder from bootstrap_templates. Paste it in public folder where **index.php** Is there. As browser runs the index.php.

***laravel suggests absolute paths.**

*master page system.

Write this in **master.blade.php** where you want any other section from **home.blade.php**.

***@yields(‘content’)** *// here content is the name of a section*

***extends(‘master’)** *//this extends the master.blade.php file. Both master and home must be in same resources folder.*

→ Inside home file, start section as:

@section(‘content’)

    <form></form>s

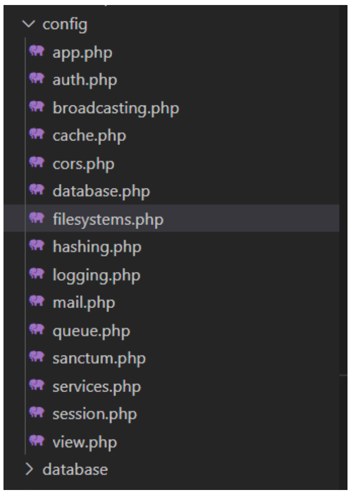
@endsection

Day 3 conclusion:

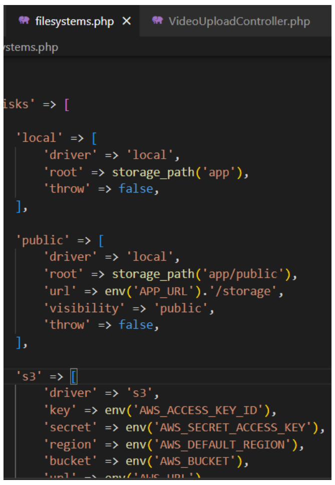
- ➔ Action can be given to any website name so the request must be from our own site, the request coming from outside should be prohibited so to protect this we write **@csrf** just below the form tag.(Cross Site Request Forgery)
- ➔ Go to laravel website -> validations to check all the validating formats for a form.
- ➔ Validating form fields: if email is with correct syntax, if any fields are empty, phone number validations.
- ➔ On day 3, front end is done with creating form and validating it.
- ➔ Created master page with **@yield** function(given above) and creating blade files to display registering forms, login forms etc.

DAY 4:

Working with input file storage and database(we're using heidiSQL)



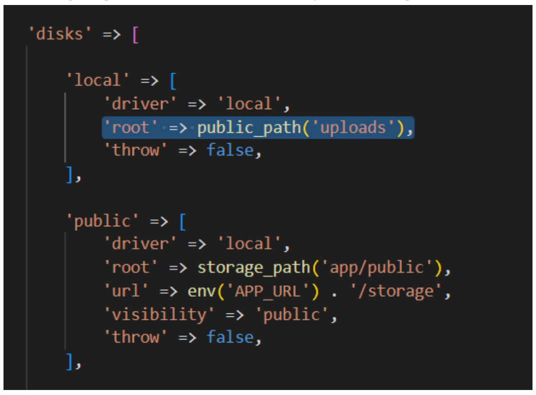
```
config
  app.php
  auth.php
  broadcasting.php
  cache.php
  cors.php
  database.php
  filesystems.php
  hashing.php
  logging.php
  mail.php
  queue.php
  sanctum.php
  services.php
  session.php
  view.php
database
```



```
isks' => [

    'local' => [
        'driver' => 'local',
        'root' => storage_path('app'),
        'throw' => false,
    ],

    'public' => [
        'driver' => 'local',
        'root' => storage_path('app/public'),
        'url' => env('APP_URL').'/storage',
        'visibility' => 'public',
        'throw' => false,
    ],

    's3' => [
        'driver' => 's3',
        'key' => env('AWS_ACCESS_KEY_ID'),
        'secret' => env('AWS_SECRET_ACCESS_KEY'),
        'region' => env('AWS_DEFAULT_REGION'),
        'bucket' => env('AWS_BUCKET'),
```

go to this filesystem.php

Changing directories for uploading files.



```
'disks' => [

    'local' => [
        'driver' => 'local',
        'root' => public_path('uploads'),
        'throw' => false,
    ],

    'public' => [
        'driver' => 'local',
        'root' => storage_path('app/public'),
        'url' => env('APP_URL') . '/storage',
        'visibility' => 'public',
        'throw' => false,
    ],
```

It says uploads goes to public folder→ uploads folder (public/uploads)

1. *Go to config / filesystem.php*
2. *Go to line 35*
3. *Change to to this: 'root' => public_path("uploads"),*
4. *Comment out line 36*
5. *Go to public folder*
6. *Create following folders: public/uploads/thumbs, public/uploads/videos*
7. *Enter into folder as: $profile_pic = $request->file('profile_pic')->store("profile_pics"); [$request is the variable of Request method in validate.]*

Model name is always singular(eloquent)

Command: php artisan make:model User

php artisan make:model User
php artisan make:model Category
php artisan make:model Contact
php artisan make:model Video

after creating a model in app/model User.php then go to homecontroller.php
public function index() {

```
$u = new User();

$u->name ="Romit";

$u->phone = "123456";

$u ->email = "abc@example.com";

$u ->password = "1234";

$u->save();
```

This saves/inserts the data inside $u in user model. Where in user.php model we have a table named as below:
Protected $table='users';

For using database, we are using a software called, Heidi sql.
after creating models, laravel suppose that same named table is there.

→Write this inside any models that you will be creating.

```
Protected $table='users';
```

→This below line disables the created at and updated at columns from the table.

```
Public $timestamps=true;
```

→this will display data with id(*primary key*) 2.

```
$u = User::find(2);
dd($u->name." ".$u->email);
```

→this says from User table display all details where id if greater than 2 and less than 5.

```
$users=User::where([

        ["id", ">", "2"],

        ["id", "<", "5"],

])->get();
```

Go to laravel site, everything is there.

Changing/editing in database with query:

```
$u=User::find(3);

$u->name='sabin';

$u->phone='9999999999';
```

This will update the phone number and name whose id is 3. (phone number is saved as varchar bc we're not performing any operation with it)

DELETING A USER:

$u=find:User(3);

$u->delete();

JOIN QUERY:

Select * from videos, users where videos.user_id=users.id;

More methods are there.

$this->belongsTo(user::class, 'id', 'user_id');

Here id is primary key, user_id is column name.

- $this->hasMany(video::class, 'user_id', 'id');

Encrypt this password:

Bcript function encrypts password.

connecting to database:

write these inside .env file: you can edit it later. We are connecting to sir's aws database.

```
DB_CONNECTION=mysql

DB_HOST=wshop.cpcm5mpbdjt1.ap-south-1.rds.amazonaws.com

DB_PORT=3306

DB_DATABASE=video_site

DB_USERNAME=video_site

DB_PASSWORD=video102030
```

1. For categories-→ admin categories.
2. Categoryblade.

day 4:

1. Created model files and did database operations in model file.
2. Validating profile_pic and video files as img/mp4 with certain kb/mb.
3. Created a file uploading field as <input type="file"> and uploaded a picture and videos.
4. That jpg and .mp4 files are stored in a file inside public folder and unique name is created for each file. *Also did inserted name, profile pics phone, descriptions etc in DB.*
5. Store the name of that files into database. No nay images and videos are directly stored in database.
6. Added categories, stored in database and displayed it in frotend from database.

Day 5(last day):

Steps to do for a fresh new project:

1. Write documents with list of features and send to client.
2. End of session , see documents sent to clients.
3. Design database.
4. Start the projects, only create web routes, all the blank pages.
5. Start with designing with forms, register, login etc.
6. Proceed to data display.

Day 5:

1. Sessions: time is logged there on visiting every page. Lost after browser closed, not the tab.
2. Cookies: file, that remembers you your email and that is saved in our device(client device).

→creating a session:

session([”fav_color”]=> ‘green’);

→route after a session:

Session()->route(‘nameofPATH’); *//name of path is the name created in web.php for a particular path. Or functions inside particular controller.*

→*data saved in session just for next page load that is known as **flash data**.*

session()->flash(‘variable', 'Task was successful!');

→when we have to display the same message everywhere, I will create a class.

→resize a image when it’s uploaded: because due to image div may be longer and shoter.

Issues with image uploads must be taken care of(dimensions).

→{{ storage::url() }} *//for uploading a file.*

→diffForHumans() *//this is carbon library, used to display **2 hours ago** message.*

→n+1 query problem.while fetching videos with user id and description,

$videos Video::orderby(‘id’, ‘desc’)->get();

we are showing user name also in blade.php file. At this point user name is not fetched in above line.

So laravel fires the query to call user name every loop. This is n+1 query problem.

So, to solve above problem, **$videos=video::with(‘user’)->orderby(‘id’, ‘desc’)->get();**

Without above line, server wil stuck if we have larger loops and laravel will run that loop many times.

→using variable in url in **web.php**.

Route………….('/video/category/{catagry_id}')………………….

→But in function it is not receiving fnction.

→Pass a category id in **index($category_id){………}**

inside **href={{ 'site.video.browse', 1 }}** *//1 here is a variable*

→sometimes we have 2 variables, in that case we use an array. We say name of variables in array as:

href={{ 'site.video.browse', [pass your array here] }}

→**findOrFail()** this function either finds from database or falis the query.

→instead of doing above line, we can create an object of a model.

→Route model binding.

→if id and password is correct in login page, store a variable in session.At logout, delete the session variable.

→middleware. Froom route →middleware→goes to controller.

Middleware is executed before goes to controller. One middleware can go to many routes.

Creating a middleware:

Cmd—php artisan make:middleware abc

In web.php

Route::get('/', [HomeController::class, "index"])->name('site.home')->middleware(abc::class);

It means we want to run this middleware before home page.

→better way to do/call middleware:

```
Route::middleware([abc::class])->groupfunction ({

        Route::get('/admin/categories', [CategoriesController::Class,
        "index"])>name("admin.categories");

});
```

Guard, providers, driver etc.

→for login authentication: inside UserController->

```
Auth::guard(‘user’)->attempt([
            “email” => $request->email,
            “password” => $request->password
])
```

→**return back()->witherrors(“**your message goes here**”) //***shows error message with a div.*