lARAVEL

**INSTALL LARAVEL VIA COMPOSER**

First, download and install composer on your system.

Then open Cli:

Second, Use composer to install laravel globally with this code “composer global require laravel/installer”

Third, Create laravel project with “laravel new project\_Name”

Fourth, navigate to the project directory with “cd project\_Name”

Five, start the server with “php artisan serve”

==================================================================

**ROUTING::**

It’s a way of making an http request to a specific url eg. form.php, contact.php

all laravel routing are defined in routes/web.php

some of it routing methods are Route::get($url, $callbackfunc), post($url, $callbackfunc), delete($url, $callbackfunc), match(['post', 'get'],

$callbackfunc), any($url, $callbackfunc), view($url,"blade\_view\_name");

To see all the available route run:: php artisan route:list

You can perform routing in 3 ways: (1) basic routing, (2) required parameters routing, (3)optional parameters routing

Most routing methods takes in 2 parameters: the url and the callback function

BASIC ROUTING:

Route::get('foo', function () {

return 'Hello World';

});

Sometimes you may need to register a route that responds to multiple HTTP request-method. You may do so using the match() method.

OR, you may even register a route that responds to all HTTP request-method using the any() method:

Route::match(['get', 'post'], 'contactpage', function () {

//this means if the request type to this url 'contactpage' was made with "post" or "get" method, execute this function. if not don't execute this method

});

Route::any('foo', function () {

//this means if any request method is made with this url 'foo' execute this function

});

Any HTML forms pointing to POST, PUT, or DELETE routes should include a CSRF token field. Otherwise, the request will be rejected

<form method="POST" action="/profile">

{{ csrf\_field() }}

...

</form>

REQUIRED PARAMETERS ROUTING::

Route::get("justice/{name}",function($namedd){

echo($namedd);

});

All required parameters should be wrapped inside "{ }" then pass it inside the callback-function as a parameter.

This will print out this url "justice/value\_you\_will\_pass\_in\_the\_url".

To Pass multiple parameters

Route::get("justice/{name}/{age}",function($named,$myage){

echo($named . $myage);

});

OPTIONAL ROUTE PARAMETERS::

means, define a url parameter with an optional value.

meaning the user may pass value to it or not. if the user don't pass any value to it, your default value pass inside the function as a parameter will be executed.

If the user pass value to it, the user value will be executed.

To create an optional route parameter, just add "?" after the parameter name and assign a default value to it in the function parameter.

Route::get("justice/{name?}",function($named="ankomah"){

echo($named);

});

==================================================================

**MIDDLEWARES**

Middleware is a way of filtering all the http request in your app and perform certain action before the app redirect to that url. Eg. you can create

a login-middleware to check if a user is login then redirect him to his profile, else if he is not login redirect him to the login page to login before he can have access to his profile page.

To create a middleware use: php artisan make:middleware MiddlewareName

you can see all your middleware in App\Http\Middleware

There are 3 types of middleware: (1)global middleware, (2) route middleware and (3)Group Middleware.

Global middleware => will run on every http request in your application. You don't need to assign it to any url because it is global once registered

route middleware => will run on only the http url the middleware is assigned to.

Group Middleware => will run on the group of rout the middleware is assigned to.

All middleware need to be registered in App\Http\kernel.php before they can be used.

To register a global-middleware simply add the middleware class name to the array $middleware variable in kernel.php. eg

protected $middleware = [

\App\Http\Middleware\MyMiddlewareClassName::class,

]

Route-MIDDLEWARE

Let say I have created this checkage middleware

class checkage{

public function handle($request, Closure $next){

if($request->age>10){

echo "age is greater than 10";

}

else{

echo "age is less than 10";

}

return $next($request);

}

}

To register a Route-middleware simply add the middleware class name to a key in the array $routeMiddleware variable in kernel.php. eg

protected $routeMiddleware = [

"agemiddleware"=>\App\Http\Middleware\checkage::class

]

Then use middleware("Middleware key") method to assign it to a http url in your app route web.php

Route::get('/{age}', function ($userage) {

return $userage;

})->middleware("agemiddleware");

TO PASS A PARAMETER TO A MIDDLEWARE:

add it to the handle method in that middleware class. eg adding $agenumber parameter:

public function handle($request, Closure $next, $agenumber)

{

if ($request->age >= 10) {

//do something

}

return $next($request);

}

Then pass the value to the above function in your route web.php using "middleware(middleware\_Key:parameter\_Value)" eg.

Route::get('/{age}', function ($age) {

return view('welcome');

})->middleware("agemiddleware:45");

Group middleware ==> is a way of assigning multiple route to a single middleware.

// Create a middleware

class checkCountry {

public function handle($request, Closure $next)

{

if ($request->country && $request->country!="ghana") {

return redirect("/home");

}

return $next($request);

}

}

// Then register it by adding it class to the kernel.php $middlewareGroups property in a key value pair.

protected $middlewareGroups = [

"checkCountry" => [\App\Http\Middleware\checkCountry::class,],

];

// Then assign multiple url route to it in your web.php using group(["middleware" => ["middleware\_key\_name"]])

Route::group(["middleware"=>["checkCountry"]],function(){

Route::get("/signup",function(){

return view("contact");

});

Route::get("/login",function(){

return view("contact");

});

});

// Then redirect to either

http://127.0.0.1:8000/login?country=ghana

http://127.0.0.1:8000/signup?country=ghana

============================================

**REQUEST**

Is use to retrieve url address and values pass into input fields or forms. eg.

"use Illuminate\Http\Request;" will give you access to all request methods

"use Illuminate\Http\Response;" will give you access to all response methods

imagine this address "http://127.0.0.1:8000/form/that"

Use $request->url() to retrieve the full address // http://127.0.0.1:8000/form/that

Use $request->path() to retrieve the path of the address // form/that

use $request->input("inpute\_Name") to retrive form values;

use $request->method(); to get the request-type the form was submitted with

Example::

If i have route of

Route::get("/form/that","retrievinput@retreiveValues");

And a form of

<form action={{"/form/that"}} method="get">

<input type="text" name="username" placeholder="enter userName" {{csrf\_token()}}/>

<br/>

<input type="submit" value="submit" />

</form>

Suppose i have this retrievinput controller

class retrievinput extends Controller

{

public function retreiveValues(Request $request)

{

//Retrive the input value

$userName = $request->input("username");

echo($userName);

//Retrive the path form/that

echo($request->path());

//Retrive the full url http://127.0.0.1:8000/form/that

echo($request->url());

echo("<br/>");

echo($request->is("that"));

}

}

======================

**Cookies**

use Illuminate\Http\Request;

use Illuminate\Http\Response;

To set a cookie..

class retrievinput extends Controller

{

//set cookie using the "Response" mehod withCookie()" and pass in the cookie("cookie\_name", "cookie\_value", "expiration\_time") methode

public function set\_cookie(Response $response){

$response->withCookie(cookie("name","justice", 1));

return $response;

}

//Retrieve cookie with "Request" method, cookie("cookie\_name") and pass in the name of the cookie

public function get\_cookie(Request $request){

$request->cookie("name");

echo($request);

}

}

//url to set above cookie

Route::get("/set\_cookie","retrievinput@set\_cookie");

//url to retrieve above cookie

Route::get("/get\_cookie","retrievinput@get\_cookie");

===========

**VIEWS**

Represent the response page the client should receive upon request

Suppose I have this "contact.blade.php" in my views directory

<?php

echo("Helloo world");

?>

To open it, just use the "view()" method and pass in the name of the file without the extension.

Route::get("/contactpage",function(){

return view("contact");

});

To pass value to "contact.blade.php" using "view()", pass in the name of the file and array of key value pairs.

Route::get("/contactpage",function(){

return view("contact", ["name" => "justice"]);

});

Then retrieve it "contact.blade.php"

<?php

echo($name);

// output = justice

?>

FORMMAT::

view("url","blade\_file\_name");

view("fileName");

view("fileName",["book"=>"Aki ola"]);

==============================

**BLADE TEMPLATE**

@yield("sidebar")

@extends("layouts.file")

@section("sidebar")

@endsection

@parent()

==================

**REDIRECTING**

Is way of redirecting a particular url to different url or a method inside a controller

Use "as" to assign a nickeName to Route urls inside an array.

Route::get("/contactpage",["as" => "contact", function(){

return view("contact");

}]);

Here i gave a nickName of "contact" to url "/contactpage"

//Then redirect a certain url to the above "contact" using redirect()->route("nickname");

Route::get("/about",function(){

return redirect()->route("contact");

});

To redirect to a method in controller user redirect()->action("controllerName@method",["parameters"]); if no parameter just ignore the parameter

Route::get("/aboout",function(){

return redirect()->action("controllerName@method");

});

==================

**SESSION**

Is a way of storing user details on the client.

To be able to create, get and delete Session. You need an instance of "Request" to access the session() method.

To create session Use:

$request->session()->put('key', 'value');

To get or retrieve session USE:

$valu = $request->session()->get('key');

To delete a session USE:

$request->session()->forget('key');

To check if session exist USE:

$request->session()->has('KEY')

Example::

use Illuminate\Http\Request; pass this to a controller and get the instance of "Request"

class retrievinput extends Controller

{

public function createSession(Request $request)

{

$request->session()->put("myName"=>"Justice");

}

public function getSession(Request $request)

{

$request->session()->get("myName");

}

public function deleteSession(Request $request)

{

$request->session()->forget("myName");

}

public function checksession(Request $request)

{

if( $request->session()->has("myName")){

//do something

}

}

}

FLASH SESSION==> is a session that will be used only once and then be deleted

// To create a flash session use $request->session()->flash("key","value");

$request->session()->flash("greetings","Good mornning sir");

// To retrieve a flash session use session("key");

{{session("greetings")}}

==================

DATABASE AND ELOQUENT MIGRATION AND MODEL

Configure your database connection in .env by adding the database name, password, ip address, users

MIGRATION is for creating table, altering table, add colums to table, and drop table. Is not for performing any CRUDE operation in the table.

MODEL == Model is for performing crud operations in the migration tables.

MIGRATION::

Every migration file represents one table in the Database.

Every migration table Class has two methods "up()" "down()".

up() is for making changes to the table.

down() is for rolling back recent changes made to the table.

To create a migration use:

//This will create a migration file that will also create a table

php artisan make:migration migration\_file\_name --create=table\_name

To create a Migration to alter another migration file:

//this will create a migration file to edit another migration file (mainly table)

php artisan make:migration migration\_file\_name --table=table\_to\_alter\_name

Creating tables in Migration::

To create a table use the "create()" method. It takes 2 parameters.

The first is the table name and the second is the Blueprint instance for creating columns.

So to create colums use the second parameter instance. example

Schema::create('tablename',function(Blueprint $table){

// Creating columns formmat $table->data\_Type\_Method("column\_Name");

//Methods== string("username"), bigIncrements(colName), integer(colName), datetime(colName), char(colName, length), text(colName),

// mediumText(colName), longText(colName), timestamp(colName)

$table->String("username");

});

The following methods are used to set additional properties of a field:

nullable()

Allows NULL values to be inserted into this column

default('default content')

Specifies the default content for this column if no value is provided

unique()

Make sure all the values in the column is unique. No duplicate content

primary()

Adds a primary key index

index()Adds a basic index

Example::

Schema::create('tablename',function(Blueprint $table){

$table->string("username")->default("justice");

});

Modifying columns::

To modify a column, just write the code you would write to create the column as if it was new, and then append the change() method to it.

Example::

Schema::table('users',function($table){

$table->string('name',6);

});

Change to==>

Schema::table('users',function($table){

$table->string('name',100)->change();

});

or to ===>

Schema::table('users',function($table){

$table->string('name',100)->nullable()->change();

});

To rename a column use renameColumn("old\_name","new\_name")

Schema::table('contacts',function($table){

$table->renameColumn('promoted','is\_promoted');

});

To drop a column use dropColumn('column\_name'):

Schema::table('contacts',function($table){

$table->dropColumn('votes');

});

To add a foreign key::

$table->foreign('user\_id')->references('id')->on('users')

here am assign a foreign key to "user\_id" in the "id" of ("users") table

// Any changes you make to your migration file will not take effect on your db table at the moment unless you run migration.

php artisan migrate

======================

**Eloquent model:::**

Every model represents a single table in your database.

To create a model use: php artisan make:model Model\_Name

To create a model with migration use: php artisan make:model Model\_Name –m

To create a model with controller and migration use: php artisan make:mode -mc

Laravel assumes that by default your table is the plural name of your model class Name. And the table id name is "id" and you have created “created\_at” time column. To override this:::

Set…

protected $table="DbtableName" ===> To set the table in your model class.

protected $primaryKey = "primarykeyName" ==> To set the name of your primary key.

public $timestamps = false; ==> To disallow the use of created at time column.

// imagine I have this model

class users extends Model

{

protected $table ="usersTable";

protected $$primaryKey = "usersid"

}

Then inside your controller::

// use the model name space

use App\users;

class querydb extends Controller

{

public function select(){

// to select value with where clause where("columnName","value")->get()

return users::where("name","justice")->get();

//to select \* from table

return users::all();

//to select using id find(idNumber) or find([idNumber,idNumber])

return users::find(3);

// to insert data

$users = new users;

$users->firstname="justice";

$users->save();

// to update data.. Below means update the table where id = 2

$users = users::find(2);

$users->firstName="justice";

$users->save();

// To delete a column

$users = users::find(2);

$users->delete();

}

}

================================================

**FORM VALIDATION**

//Doing validation in a controller function

public function store(Request $request)

{

$request->validate([

'email' => 'required|unique:posts|max:255',

'frist\_name' => 'required',

]);

//displaying the errors inside signup.blade.php

$errors 🡺 represent the instance of all errors message by default in laravel

//check if errors is greater than 0

@if (count($errors) > 0)

<ul>

// loop through the error and displays it

@foreach ($errors->all() as $error)

<li>{{ $error }}</li>

@endforeach

</ul>

@endif

OR

To display error under each input filed:

// $message represent the instance of error messages

<input type="email" name="email" />

@$error("email)

<p> {{$message}} </p>

@enderror

================================================

LARAVEL SINGLE ITEMS CLICK OR DELETE HACK

//Button to delete Single items

@foreach($data as $resdata)

<tr>

<td>{{$resdata["restaurant\_id"]}} </td>

<td> {{$resdata["restaurant\_name"]}}</td>

//single item delete link

<td> <a href="/delete/{{$resdata["restaurant\_id"]}}" > Delete </a Delete></td>

//if the first and second item is clicked the above url will be:

http://127.0.0.1:8000/delete/1

http://127.0.0.1:8000/delete/2

</tr>

@endforeach

// Route parameter with unique id

Route::get("/delete/{id}","RestoController@delete");

// controller to find the id of the item and delete it

class RestoController extends Controller{

public function delete($id){

$dele =restaurant::find($id);

$dele->delete();

}

}

=====================================================================

To hash a password And set Session:::

//inside controller use the Crypt api in config/app.php;

use Illuminate\Support\Facades\Crypt;

Class register extends controller{

//Signup method to hash password and set Session

public function register(Request $req){

// “User” is a model (database table)

$user=new User;

//hash the password

$hashp = Crypt::encrypt($req->input("password"));

$user->password=$hashp;

$user->save()

//set session

$reg->session()->put("user", $req->input("username"));

}

//Login method to check password and set session

public function login(Request $reg){

//find the email in the db

$user = User::where("email, $req->input("email"))->get();

// decrypt password and check if it's equal to what is hashed stored in db

if(Crypt::decrypt($user[0]["userpassword"]) == $req->input("userpassword")){

//set sesion

req=>session()->put("user",$req->inpute("username"));

//return to user dashboard

}

else{

//return to login

return redirect("/login");

}

}

}

================================

**LARAVEL AUTHENTICATION**

Authentication in laravel is a default login and registration laravel come with by default.

To generate it use:: php artisan ui:auth

The above will generate a login, registration and forget password views as well as migration with controllers for performing all the default registration.

========================

**LARAVEL SHEDULER**

Scheduler in laravel means scheduler some piece of code to be run on specific time.

first create a commander by using: php artisan make:command justicescheduler

you can see it in app/console/commands/

Then open the command file example above as justicescheduler.php and add name and description for your command in the

protected $signature ='jucicecommand'

protected $description ="my new command".

Then write the code you want to run on timely bases in the handle() method.

AFTER THAt, YOU need to register it in app/console kernel.php file. Just add your command name to command() method and the minute you want to run the command inside the schedule() method.

protected function schedule(Schedule $schedule)

{

$schedule->command('justicecommand')->everyMinute();

}

Then you can run it by using: php artisan schedule:run this will run your scheduler in base on the time specified in the kernel.php file.

you can also use php artisan list to see the command.

================================================================

**SEEDER**

Seeders are used to fill database table with dummy fake data.

//All the seeders will be placed in “database/seeders” directory

//all seeder class will have a run() method which will be executed whenever you seed the that class.

// To create a seeker use: “php artisan make:seeder seeder\_Name”

Eg. Php artisan make:seeder Userseeder

// now inside above created seeder class, import the model you want to fill with dummy data and specify how many in the factory() method.

<?php

namespace Database\Seeders;

use Illuminate\Database\Seeder;

use App\Models\User;            //import this

class UserSeeder extends Seeder

{

    /\*\*

     \* Run the database seeds.

     \*

     \* @return void

     \*/

    public function run()

    {

     User::factory(10)->create();  // insert the number of random data to create in factory() method

    }

}

//Now you can specify the model and the columns to fill in “database/factories/userFactory.php”

//Then inside “database/seeders/DatabaseSeeder” class add the above created seeder to the call method. All your created seeders class should be added in there

    public function run()

    {

        // \App\Models\User::factory(10)->create();

        $this->call([ //add all your seeder class into this call([]) array method

            UserSeeder::class

        ]);

    }

//Finally to seed a specific seeder class call php artisan db:seed –class=Seeder\_class\_name\_here

php artisan db:seed --class=UserSeeder

================================================================

**LARAVEL API**

By default laravel comes with two routing service. “Web” and “api”.

These two services are provided in “providers/RouteServiceProvider.php” boot() function.

Route “web” is the normal routing service.

Route “api” is what you should use when building laravel api.

So always delete below “Route Web” method when building api to prevent the use of web route. Or simply comment it.

    public function boot()

    {

        $this->configureRateLimiting();

        $this->routes(function () {

            Route::prefix('api')

                ->middleware('api')

                ->namespace($this->namespace)

                ->group(base\_path('routes/api.php'));

          /\*  Route::middleware('web') //comment it out

                ->namespace($this->namespace)

                ->group(base\_path('routes/web.php')); \*/

        });

    }

The “prefix(‘api’)” define a custom prefix for all api’s. And you can change it to whatever prefix you want. This is what I mean <http://127.0.0.1:8000/api/>

Then install passport for api authentication 0Auth

**Laravel Passport**

It’s used to secure api’s in laravel with 0auth2 api authentication method.

Before installing, if your laravel app server is running, please stop and install the passport…

// First install it with

“composer require laravel/passport”

Once the installation is complete, a new migration file containing the tables needed to store clients and access tokens will be generated for your application. Run the following command to migrate your database:

“php artisan migrate”

Next, to create the encryption keys needed to generate secured access tokens, run the command below:

“php artisan passport:install”

ON YOUR CLI, IT WILL GENERATE TWO CLIENT ID’S COPY IT AND ADD IT TO THE BOTTOM OF YOUR .ENV FILE

CLIENT\_1 = Client\_secret\_key\_here

CLIENT\_2 = Client\_secret\_key\_here

Then in your “App\Models\User” add the passport “use Laravel\Passport\HasApiTokens” then add it to “use Notifiable” property in the User class. Below is what I mean. (If you get error on this step, just restart visual studio code. If the server too is running, terminate it and open new cli to restart it)

One of the benefits of this trait is the access to a few helper methods that your model can use to inspect the authenticated user’s token and scopes

use Laravel\Passport\HasApiTokens; //include this

class User extends Authenticatable

{

    use Notifiable, HasApiTokens; //include HasApitokens

Next, you should call the Passport::routes() method within the boot method of your App\Providers\AuthServiceProvider. This method will register the routes necessary to issue access tokens and revoke access tokens, clients, and personal access tokens:

  public function boot()

    {

        $this->registerPolicies();

        Passport::routes();  //add this

    }

After registering Passport::routes()above, Laravel Passport is almost ready to handle all authentication and authorization processes within your application.

Finally, in your application's config/auth.php configuration file, you should set the driver option of the api authentication guard to passport. This will instruct your application to use Passport's TokenGuard when authenticating incoming API requests:

'guards' => [

        'web' => [

            'driver' => 'session',

            'provider' => 'users',

        ],

        'api' => [

            'driver' => 'passport', // set this to passport,

            'provider' => 'users',

            'hash' => false,

        ],

    ],

You are done installing laravel passport and is ready to use. Run “php artisan route:list” to see passport available urls

//Then create a seeder to add some fake data to your “users” table.

//Then make a post request to <http://127.0.0.1:8000/oauth/token> in Postman and include below body parameters to login a user and generate a token for the user… after it click on send and if successful, it will generate a token for you

Username = “one of the registered users email in users table”

Password = “the above user password. If you used seeder, check the database/userFactory.php there is a some password comment check it. It mostly password or secret”

grant\_type = password

scope = \*

client\_id = 2

client\_secret = copy the client 2 secrete you have saved in your .env file

if everything is successful, Passport will generate a token for you, your users need to include that token in every api request like todolist api.

For QUICK TIPS === THINGS SHOULD BE…..

Your rout.api should be

// middlewaare('auth:api') below is defualt laravel api middlware to allow user to login only when the user is authenticated with token

Route::middleware('auth:api')->group(function () {

    // Route to show the login user details

    Route::get("/user", "userController@userdetails");

});

KERNEL.PHP SHOULD BE

   protected $routeMiddleware = [

        'auth' => \Illuminate\Auth\Middleware\Authenticate::class, // this authentication class need to be adjusted to this

        'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,

        'cache.headers' => \Illuminate\Http\Middleware\SetCacheHeaders::class,

        'can' => \Illuminate\Auth\Middleware\Authorize::class,

        'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,

        'password.confirm' => \Illuminate\Auth\Middleware\RequirePassword::class,

        'signed' => \Illuminate\Routing\Middleware\ValidateSignature::class,

        'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,

        'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,

    ];

}

Userscontroller.php should be

<?php

namespace App\Http\Controllers;

use App\Models\User;

use Illuminate\Http\Request;

class userController extends Controller

{

    //Get login userdetails

//In postman or client app, pass the token that will be generated during user login at <http://127.0.0.1:8000/oauth/token> as a hearder here example:

// Authorization: “Bearer user\_token\_eyJ0eXAiO”

        function userdetails(Request $request){

        return $request->user();

    }

    //REGISTER USER // No need to send any headers/body token in postman or client app during registration

    public function register(Request $request)

    {

        //validate form input

        $this->validate($request, [

            "name" => "required|min:3",

            "email" => "required|email",

            "password" => "required|min:6"

        ]);

        $user = User::create([

            "name" => $request->name,

            "email" => $request->email,

            //hash the password and insert it into db "password" column

            "password" => bcrypt($request->password)

        ]);

        return response()->json(["Success" => "Registration Successful"], 200);

    }

}

================================================================

**FIXING CORS ISSUE**

// Navigate to your project directory in CLI and install the laravel cors package with:

“composer require fruitcake/laravel-cors”

//check your composer.json to see if you have any version of the above cors like: "fruitcake/laravel-cors": "^2.0",

//Next open “app/http/kernel.php” and add the cors middleware class “\Fruitcake\Cors\HandleCors::class, to the  **protected** $middleware array.

  protected $middleware = [

        // \App\Http\Middleware\TrustHosts::class,

        \App\Http\Middleware\TrustProxies::class,

        \Fruitcake\Cors\HandleCors::class,

        \App\Http\Middleware\PreventRequestsDuringMaintenance::class,

        \Illuminate\Foundation\Http\Middleware\ValidatePostSize::class,

        \App\Http\Middleware\TrimStrings::class,

        \Illuminate\Foundation\Http\Middleware\ConvertEmptyStringsToNull::class,

        \Fruitcake\Cors\HandleCors::class,   // ADD THIS LINE HERE

    ];

//Next you need to publish the configuration file to your project directory with:

php artisan vendor:publish --tag="cors"

//A new file cors.php should be added to “config/cors.php” so open it and make sure the “paths”, “allowed\_methods”, “allowed\_origins” is like //below. You are done, that should fix everything.

    'paths' => ['api/\*', 'sanctum/csrf-cookie', "\*"],

    'allowed\_methods' => ['\*'],

    'allowed\_origins' => ['\*'],

    'allowed\_origins\_patterns' => [],

    'allowed\_headers' => ['\*'],

    'exposed\_headers' => [],

    'max\_age' => 0,

    'supports\_credentials' => false,

];

===================================

Validate input values

public function register(Request $request){

//Validate form inpute. the validate($request\_object, ["Form\_input\_Name"=>validation rules])

$this->validate($request, [

"name"=>"required|mind:3",

"email"=> "required|email|unique:users",

"password"=> "required|mind:6"

]);

}