Framework

php artisan route:cache

php artisan cache:clear

php artisan config:clear

Top 10 PHP frameworks

A PHP framework is a platform which allows a web developer to develop the web application. In simple words, it provides a structure to develop web application. These frameworks save lots of time, stop rewriting the repeated code and provide rapid application development (RAD). PHP frameworks help the developers to rapid development of application by providing the structure.

Following a list of top 10 PHP frameworks is given below:

1. **Laravel Framework**
2. **CodeIgniter Framework**
3. **Symfony Framework**
4. **Zend Framework**
5. **CakePHP Framework**
6. **Phalcon Framework**
7. **Yii Framework**
8. **Slim Framework**
9. **FuelPHP Framework**
10. **PHPixie Framework**

Step :1

Mvc : mvc project

Framework : **Laravel**, symfony, codeigniter

CMS : WORDPRESS / MAGENTO

1. **What Is Laravel**

- **PHP Framework**
- **For Developing web app and API**
- **Free Of Cost**
- **Modern Framework and easy to use**
- **The Most Used Framework in PHP**

2. **History and Version**

- **First Release on June 2011**
- **Current Version 12.0**
- **Developer Name : Taylor Otwell**
- **Written IN php**

3. **Why use Laravel**

- **Strong Command Line Support / CMD COMMAND**
- **Large Community / doc**
- **Regular Updates**
- **Fast and simple**

==========

4. Requirement

- **PHP - 8.1 version of xampp or greater than**
- **Composer**

1) Xampp:

X cross platform

A apache

M mysql

P perl

P php

First check your **xampp** version

its required 8.0> upgrade version so instal

==========

2)

First Download **Composer** in website

getcomposer.org

2 method

1. **use Manually Windows Installer (Preferable)**
2. **By cmd Command-line installation**

1)

https://getcomposer.org/download/

Than instal on **xampp/php/php.exe**

==========

2) By cmd Command-line installation

php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"

php -r "if (hash_file('sha384', 'composer-setup.php') === '55ce33d7678c5a611085589f1f3ddf8b3c52d662cd01d4ba75c0ee0459970c2200a51f492d557530c71c15d8dba01eae') { echo 'Installer verified'; } else { echo 'Installer corrupt'; unlink('composer-setup.php'); } echo PHP_EOL;"

php composer-setup.php

php -r "unlink('composer-setup.php');"

==========

3)

Than go CMD & check version

Enter==>composer -v // ok

==========

4) Installation of Laravel 11

There are two TYPE

1. **Laravel Installer**

Composer Create-ProjectJust Only Run one time in your pc

1)composer global require laravel/installer note: one time only

Then you can create project by belove command every time

2)laravel new project_name

Note : proper version required for composer & xampp

Now open CMD & GO xampp

cd xampp/htdocs/ourfoldername

2. composer create-project laravel/laravel laravel_project

After Installation Run Project and default start in browser (localhost:8000)

Enter==> htdocs-> laravel_test- >php artisan serve

==========

Output : welcome

view page : resource/view/welcome.blade.php

url set: routes/web/ set route

==========

5) now check INDEX PAGE OF LARAVEL 2 way

a) open in xampp/project_name/public/ //open your index page

b) localhost:8000 //default create port for laravel

If want to first change in Laravel

1)Project/rourtes/web.php set our routes // create routes

2)Project/resources/view/mypage.blade.php // create view page

==========

6) Folder Structure of Laravel

App

It is the application folder and includes the entire source code of the project. It contains events, exceptions and middleware declarations, models and in http it contains controllers also. The app folder comprises various sub folders as explained below −

App/model ==== model.php page banana

App/http/controller/ controller.php page banana

App/http/Middleware/ Routes ke middleware set karne hote he

Console

Console includes the artisan commands necessary for Laravel. It includes a directory named Commands, where all the commands are declared with the appropriate signature. The file Kernal.php calls the commands declared in Inspire.php.

We can also create custom command

Events

This folder includes all the events for the project.

Events are used to trigger activities, raise errors or necessary validations and provide greater flexibility. Laravel keeps all the events under one directory. The default file included is event.php where all the basic events are declared.

Exceptions

This folder contains all the methods needed to handle exceptions. It also contains the file handle.php that handles all the exceptions.

*** Http --- Model , Controller and Middleware**

The Http folder has subfolders for controllers, middleware and application requests. As Laravel follows the MVC design pattern, this folder includes models, controllers and views defined for the specific directories.

The Middleware sub-folder includes middleware mechanism, comprising the filter mechanism and communication between response and request.

The Requests sub-folder includes all the requests of the application.

Jobs

The Jobs directory maintains the activities queued for Laravel application. The base class is shared among all the Jobs and provides a central location to place them under one roof.

Listeners

Listeners are event-dependent and they include methods which are used to handle events and exceptions. For example, the login event declared includes a LoginListener event.

Policies

Policies are the PHP classes which include the authorization logic. Laravel includes a feature to create all authorization logic within policy classes inside this sub folder.

Providers

This folder includes all the service providers required to register events for core servers and to configure a Laravel application.

Bootstrap

This folder encloses all the application bootstrap scripts. It contains a sub-folder namely cache, which includes all the files associated for caching a web application. You can also find the file app.php, which initializes the scripts necessary for bootstrap.

* Config

The config folder includes various configurations and associated parameters required for the smooth functioning of a Laravel application. Various files included within the config folder are as shown in the image here. The filenames work as per the functionality associated with them.

We have lots of configuration in this folder like Database / Session / mail

Database

As the name suggests, this directory includes various parameters for database functionalities. It includes three sub-directories as given below −

- Seeds − This contains the classes used for unit testing databases. Means fake data
- Migrations − This folder helps in queries for migrating the database used in the web application.
- Factories − This folder is used to generate a large number of data records.

we can work all database and table work by files , you can create and drop table and also you can create fake data and seeding data

database/migration // sql tables file create

database/seeders

*Public

It is the root folder which helps in initializing the Laravel application. It includes the following files and folders −

- .htaccess − This file gives the server configuration.
- javascript and css − These files are considered as assets // all theme folder.
- **index.php** − This file is required for the initialization of a web application.This is first file in laravel load when project Run

*Resources : We load public file of theme and

Resources directory contains the files which enhances your web application. The sub-folders included in this directory and their purpose is explained below −

- assets − The assets folder includes files such as LESS and SCSS, that are required for styling the web application.
- lang − This folder includes configuration for localization or internalization.
- **views** − Views are the HTML files or templates which interact with end users and play a primary role in MVC architecture.We all our HTML page in this folder

resources/ view all project view page

Routes :

The routes directory contains all of the route definitions for your application. By default, several route files are included with Laravel: web.php, api.php, console.php, and channels.php.

Routes/web.php all pages routes/url set in this page

Storage

This is the folder that stores all the logs and necessary files which are needed frequently when a Laravel project is running. The sub-folders included in this directory and their purpose is given below −

- app − This folder contains the files that are called in succession.
- framework − It contains sessions, cache and views which are called frequently.
- Logs − All exceptions and error logs are tracked in this sub folder.

Tests

All the unit test cases are included in this directory. The naming convention for naming test case classes is camel_case and follows the convention as per the functionality of the class.

Vendor

Laravel is completely based on Composer dependencies, for example to install Laravel setup or to include third party libraries, etc. The Vendor folder includes all the composer dependencies.

In addition to the above mentioned files, Laravel also includes some other files which play a primary role in various functionalities such as GitHub configuration, packages and third party libraries.

.env : All the laravel credential in this files and very importance // database connectivity

APP_KEY=base64:fmEr4jlgDbcM5mL9pNxZ6pjU9Y8IOTHYLuaUpYpr2Zo=

DB_CONNECTION=mysql

DB_HOST=127.0.0.1

DB_PORT=3306

DB_DATABASE=hospital

DB_USERNAME=root

DB_PASSWORD=

Composer.json : All the configuration of laravel in this files and all dependency

Package.json : when we developing api and use in front and developer likes React and Angular and etc.. that times it used

Que : Where is laravel version packages and dependency and also how change ad after that what command fire in laravel

==========

Migration:

Migrations are like version control for your **database**, allowing your team to define and share the application's database schema definition. If you have ever had to tell a teammate to manually add a column to their local database schema after pulling in your changes from source control, you've faced the problem that database migrations solve.

The Laravel **Schema facade** provides database agnostic support for **creating** and **manipulating** tables across all of Laravel's supported database systems. Typically, migrations will use this facade to create and modify database tables and columns.

optional : php artisan config:Cache

cache reconfig for our db connection if error occurs

==========

php artisan make:migration create_posts_table

php artisan make: migration create_posts_table --create=posts // with structure

Php artisan migrate

```
public function up()
  {
     Schema::create('clients', function (Blueprint $table) {
       $table->id();//$table->increments('id');
                    or
$table->id('custome_id');  // you can also add custom id

// it provide default created_at/ updated_at column by default
       $table->timestamps();
```

$table->integer('votes');

$table->string('username'); // default size 255

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

$table->char('name', 100)->index();

$table->char('name', 100)->unique();

$table->text('description1');

$table->longText('description');

$table->timeTz('sunrise', $precision = 0);

$table->dateTimeTz('created_at1', $precision = 0);

$table->double('amount1', 8, 2);

$table->float('amount', 8, 2); $table->enum('status',['Block','Unblock'])->default('Unblock');

$table->bigInteger('mobile');

$table->unsignedBigInteger('cate_id'); // foreign key declare $table->foreign('cate_id')->references('id')->on('categories');

$table->foreign('user_id')->references('id')->on('users')->onDelete('cascade')

Or

ss

Create all table in phpmyadmin

php artisan make:migrate

==========

Add column after create table

php artisan make:migration create_countries_table

php artisan make:migration add_columns_to_countries

Then its create again new migration file for customer then add remaining field in new generated migration file

$table->string('country',60)->nullable()->after('address');

$table->string('state',50)->nullable()->after('state');

php artisan migrate:make **add_columns_to_countries**

php artisan migrate

==========

If you don't want migration table & want to create own table by yourself directly in phpmyadmin

1. **table name with s users**
2. **fix primary key name (id)**
3. **fix 2 column 1)created_at 2)updated_at**

==========

php artisan migrate:status : like to see which migrations have run thus far

php artisan migrate --force

php artisan migrate:rollback // only one roleback

php artisan migrate:rollback --step=4

php artisan migrate:reset : command will roll back all of your application's

migrations

php artisan migrate:refresh :command will roll back all of your migrations and then execute the migrate command. This command effectively re-creates your entire database: // rollback and fresh migration

php artisan migrate:fresh :The migrate:fresh command will drop all tables from the database and then execute the migrate command:

==========

Crud command

php artisan make:migration create_posts_table //table create s wala

php artisan make:model post // model create s biger

php artisann make:controller postController --resource// control creat

All in one command

php artisan make:model User --c --resource --migration

php artisan make:model --migration --controller test --resource

php artisan make:model Todo -mcr

1. **create user model**
2. **create users table**
3. **create userController**

==========

Note : If not use that types of rule means

If table name emp and model name also emp then that types of case you have to configure some manually

Add in model page class :

public $table="emp"; // if table & model name not same then use this

public $primarykey="emp_id"; // if want custom primary key in table

//If don't want then add : created_dt and update_dt

public $timestamps=false in user model class

==========

For database connectivity you can use any DB software from

Config/database.php ===

Default mysql / **sqlite / pgsql / sqlsrv /**

DB Connectivity: Go in .env file set DB name / Then migration new connectivity new Catch

==========

The End

==========

php artisan make:migration users table name always with s

Php artisan make:model user model without s but same name

php artisan make:model user -m // make model with migration file

==========

==========

7) Create First File (Make First Change and create File In laravel)

Public/ Resources/ Views

All HTML pages in this folder

Welcome.blade.php main First load on laravel run./ Default file

How can we do ?

Just create own file must myname.blade.php

Routes/web.php Main route file for run our pages

```
Route::get('/', function() {
        Return view('myname');
})
```

==========

Que: Can we change views folder name yes or no and if yes then how ?

12) Laravel Blade Template

What is blade template

The Blade is a powerful templating engine in a Laravel framework.

The blade allows to use the templating engine easily, and it makes the syntax writing very simple.

The blade templating engine provides its own structure such as

conditional statements and

loops.

To create a blade template, you just need to create a view file and save it with a .blade.php extension instead of .php extension.

The blade templates are stored in the /resources/view directory. The main advantage of using the blade template is that we can create the master template, which can be extended by other files.

Also You can make common header and footer also

Why Blade template ?

- **Displaying data**
  If you want to print the value of a variable, then you can do so by simply enclosing the variable within the curly brackets.
  Syntax
- **echo “Hi hello”**

1. {{“hi hello”}};

In blade template, we do not need to write the code between **<?php echo $variable; ?>**. The above syntax is equivalent to **<?= $variable ?>**.

Blade Template Expression

{{}} echo

{{--}} Comment in laravel by blade templating / // or /* */

We can directly use php code without in user.blade.php <?php ?> syntex

<h1>{{10+10}}</h1>

<h1><?php echo 10+10; ?></h1>

We can direct use php function leke below

<h1>{{count($users)}}</h1>

Blade Conditional Directives

@php @endphp

@if , @elseif ,@else and @endif

@unless , @endunless // inverse of if / opposite of if

@isset @endisset

Condition

<?php $name="nagar"?>

@if($name=="Raj")

<h1>Hi my name is {{$name}}</h1>

@elseif($name=="Mahesh")

<h1>Hi my name is {{$name}}</h1>

@else

<h1>Unknown</h1>

@endif

- **Ternary operator**
  In blade template, the syntax of ternary operator can be written as:

1. {{ $variable **or** 'default value'}}

The above syntax is equivalent to **<?= isset($variable) ? $variable : ?default value? ?>**

Blade Looping Directives

@for and @endfor

@while and @endwhile

@foreach and @endforeach

@break @continue

==========

For Including @include('layouts.frontend.login')

For raw PHP @php @endphp

Layout Blade Directives

@include

@yield directive is used to display the content of a given section

@section and @endsection directives define a section of content

@extends blade directives specify which layout the child view should “inherit”

@stack render the complete stack content , pass the name of the stake

@push and @endpush is used to push the stack

Header.blade.php page

```
<head>
        @stack(‘title’)
</head>
```

View.blade.php

```
@push(‘title’)
<title>Home</title>
@endpush
```

For and foreach loop

```
<p>We can use conditional loop in blade template </p>

@for($i=1;$i<=10;$i++)
<h4>{{$i}}</h4>
@endfor

<p>We can use Foreach loop in blade template </p>

<?php $data=['sam','raj','mahesh'];?>

@foreach($users as $d)
<h4>{{$d}}</h4>
```

@endforeach

Interview Question

1. What is Blade Template in Laravel

==========

8) Routing

What is Routing

Routing Method

How make Routing

How can I pass data in routing

Anchor Tag

Redirect

Routing : Mapping our laravel page in specific url

Routes/web.php have default routing for welcome.blade.php

Route

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

There are two thing in it

1. Page name : welcome page name
2. Url : Default route routing '/'

Now create one page and provide route : about.blade.php

Now create route for call about page in web.php:

```
Route::get('/about', function() {
        Return view(about);
})
```

Note : You can also use Small syntax for route :

```
Route::view("about_url","aboute_page");
```

Routes Method

Any ===> any use & work all method

Get ===> use for views

Post ===> use for store / insert

Put / Patch ===> store / update

Delete ===> delete

How can I pass data in routing

```
Route::get('/{name}', function($name) {
        echo $name;
Return view(about);
})
```

Now You can also pass direct value in page :

```
Route::get('/{name}', function($name) {
Return view(about,['name'=>$name]);
})
```

```
<h1>{{$name}}</h1>
```

Anchor Tag in laravel : As simple as usual

```
<a href="about">About </a>
```

Redirect Page : You can also default redirect in project

```
Route::get('/', function() {
        Return redirect("about");
//Return view(about);
})
```

Que : What is api.php in the routes folder and what use for it ?

==========

Template Integration :

Step 1 : Download Template

Step: 2 : copy all css & assets folder in **Public folder like website or admin**

website : all website folder

admin: all admin

Step: 3 add all view page in resources/view like website / admin

1)Also convert in blade.php store in resources/website/

2)also create Layouts folder in frontend

1)Layout/header.blade.php

HEADER CODE

2)Layout/main.blade.php

@include(‘website.layout.header')

@yield('main_container')

@include('website.layout.footer')

3)Layout/footer.blade.php

FOOTER CODE

3)Also add layouts like in all blade.php

@extends(‘website.layout.main'); remove header & footer portion

Add below section in main content

@section('main_container')

--------CODE--------

@endsection

4) all add {{url('website/assets/css.style.css')}} in all path of design

Its take you in public folder

5) then make Routes in web.php

6) set link in header page as per Routes {{url('/about')}}

==========

Layout Blade Directives

@yield directive is used to display the content of a given section

@section and @endsection directives define a section of content

@extends blade directives specify which layout the child view should "inherit"

@stack render the complete stack content , pass the name of the stake

@push and @endpush is used to push the stack

==========

9) Controller : Central units of any MVC framework.

What is Controller

Controller fetch the data from model and send to the view so its meadiater of Model and View

Controllers can group related request handling logic into a single class

All router linked with controller

All Logic in controller

Type of controller and Make Controller

- Basic Controller

1. **Php artisan make:controller userController** // app/http/controller/users.php

use Illuminate\Http\Request; also add library in control page for request data

Route::get('/mycontrol',[**userController**::class, 'myfunction']);

- Single action controllers only support one action / -invokable /__construct

2. Php artisan make:controller invokableController --invokable

Then yun only access direct function from class

=> Route::get('/data',singleController::class)

- Resource Controllers // with all crud functionality

3. Php artisan make:controller userController --resource

main : Php artisan make:controller userController --resource --model=User

Route::get('/mycontrol',[Mycontroller::class, 'index']);

Route::resource('/user',**userController**::class); // auto call index by URL

==========

All action handled by Resource Controller

Actions Handled By Resource Controller

Method	URI	Action	Route Name
GET	/photos	index	photos.index
GET	/photos/create	create	photos.create
POST	/photos	store	photos.store
GET	/photos/{photo}	show	photos.show
GET	/photos/{photo}/edit	edit	photos.edit
PUT/PATCH	/photos/{photo}	update	photos.update
DELETE	/photos/{photo}	destroy	photos.destroy

There are two method to create controller

1. By cmd (prefer this save your times)
2. By create new file in controller and code copy paste and change class name

Make function in Controller : Just normal way as per oops concept

```
Public function index()
{
        Return view('about');
}
```

Call controller from Routing

There are two method in laravel

1. Old in 6 and 7
2. New in laravel 8

We go **web.php**

USE IN 7 AND OLD version so its get error in 8 version

//Route::get(“path”,”controller file@function name”);

//Routes::get(“/users”,”Users@index”) ; call function by Users Control

Use New 8 Version

Add controller path fist in router

Use App\Http\Controllers\Users;

Route::get('/mycontrol',[Mycontroller::class, 'myfunction']); // call function in Users CONTROL

Pass Params with URL

User.php controller

```
Function index($name)
{
        echo $name;
echo “welcome Users Control”;
}
```

Web.php

Route::get("users/{name}",[Users::class, 'index']); // call function in Users CONTROL

Que: why controller call method change in 8

==========

10) Laravel View

Public / Resources / views / welcome.blade.php

What is view : mvc part : all html part which display in websites

Make View: Just Create file in views folder

Call View :

There 2 way

1. By direct Routing

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

return view(user);

})

Or

Route::view(“user”,”user”); // Limitation don’t pass value in this method

2. By Controller

Create Controller : **php artisan make:controller UsersController**

```
Class UsersController {
    Function loadview()
{
    Return view(user);
}
}
```

Call controller in web.php

```
Use App\Http\Controllers\UsersController;

Route::get(“user”,[UserController::class, ‘loadview’]);
```

Pass data(values) in View

```
Route::get(‘/user/{name}’, function($name) {
return view(“user”,[name=>$name]);
})
```

User.blade.php

<h1>{{name}}</h1>

Que:

Why don't I make a view with CMD ?

Before I run code and run the view , can I check if the view is available or not ? Then how ?

12) Anchor tag and Button

It's always first take Route::get() method

< a href="{{ url('/register') }}"> Register its take http/:localhost:8000/

Wants Custom Routes

Route::get('/customer/create', [controller::class,'create'])->name('customer.create');

Now add above routes

 Register Here

Now want to add route with id // for delete or edit

$data->cust_id]) }} "> Register Here

11) Laravel Component

What is component

Components are the re-usable section or layout which we can use inside laravel application. In laravel there are two methods to writing a components: class based components and anonymous components.

Much pretty to use as function also but its advance of function.

Function have limited functionality

Because we can add php,html and we can add database connectivity code also in components.

Best example u can say : Header - Reusable things

Make Component

php artisan make:component Header

When we run this, it creates 2 files into setup. Let's see what those files are.

make:component command creates a view template file as well as a Component class file. View layout file we will find inside **/resources/views/components/header.blade.php**. Along with this view file we also have a component class file. Class file we can find inside **/app/View/Components/Header.php**

Use Component

<x-header/>

Pass data in Component

<x-header data="hello pass value in component - from user"/>

app/view/component/Header.php (add below code on it)

```
public $title;
public function __construct($data)
{
```

```
    $this->title=$data;
  }
```

Interview Question

13) Laravel HTML Form

Laravel provides various inbuilt tags to handle HTML forms easily and securely. All the major elements of HTML are generated using Laravel. To support this, we need to add an HTML package to Laravel using composer.

What is CSRF in laravel form : Security Tokens

CSRF refers to **Cross Site Forgery attacks** on web applications. CSRF attacks are the unauthorized activities which the authenticated users of the system perform. As such, many web applications are prone to these attacks.

Laravel offers CSRF protection in the following way −

Laravel includes an in built CSRF plug-in, that generates tokens for each active user session. These tokens verify that the operations or requests are sent by the concerned authenticated user.

- CSRF is implemented within HTML forms declared inside the web applications. You have to include a hidden validated CSRF token in the form, so that the CSRF protection middleware of Laravel can validate the request. The syntax is shown below −

```
<form method = "POST" action="/profile">
  {{ csrf_field() }}  OR    @csrf
  ...
</form>
```

Make Controller

Route View and Post

Get Form Data from in to controller function in Json format

Web.php

```
Route::post("getdata",[Formcontroller::class,'getData']);
Route::view("login","form");
```

login.blade.php

```
<form action="getdata" method="POST">

<input type="text" name="userid" placeholder="Enter User Id"><br><br>
<input type="password" name="password" placeholder="Enter User Password"><br><br>
<input type="submit" name="submit">

</form>
```

Formcontroller.php

```
function getData(Request $request)
        {
                Return $request->input();
        }
```

Interview Question

16) Laravel Start with DB

Laravel connect database in two way

1. Config database .env and import DB class
2. Database connectivity by model

Crud In laravel

There are two to create crud in laravel

1. By Query Builder / DB CLASS
2. By Model / ORM

Config Database

Find **.env** file and config database connectivity

DB_CONNECTION=mysql

DB_HOST=127.0.0.1

DB_PORT=3306

DB_DATABASE=laravel8

DB_USERNAME=root

DB_PASSWORD=

Note: If you don't find .env file then just search .env.example file and rename .env

Checkout Database

Above all configuration done from **config/database.php**

Import DB class in Controller

// ADD FOR USE ANY DATABASE QUERIES

use Illuminate\Support\Facades\DB;

Fetch Data From MySql

Function user()

{

$data= DB::select("select * from users");

return view('viewuserdb',['mydata'=>$data]);

}

user.blade.php

@foreach ($mydata as $items)

<tr>

<td>{{$items->uid}}</td>

<td>{{$items->unm}}</td>

<td>{{$items->pass}}</td>

<td>{{$items->email}}</td>

<td>{{$items->mobile}}</td>

</tr>

@endforeach

==========

17) Laravel Model with DB

What is model

Models are class based php files

Laravel Includes Eloquent ORM (Object-Relational Mapping) that makes it enjoyable to interact with your database

A Model in Laravel 8 provides **an abstraction for working with a database table with a high-level API**. ... Model events are simply hooks into the important points of a model's life cycle which you can use to easily run code when database records are saved, updated or deleted.

A Model is **basically a way for querying data to and from the table in the database**. Laravel provides a simple way to do that using Eloquent ORM (Object-Relational Mapping). Every table has a Model to interact with the table

Before Use you have to know that :

Map Database table with Class Name means same name of table with S & model name without S so model automatically connect table

Exa	DB Table(Plural)	Model Name(simular)
	users	user
	employees	employee

==========

Note : If not use that types of rule means

If table name emp and model name also emp then that types of case you have to configure some manually

Add in model page class :

public $table="emp"; // if table & model name not same then use this

public $primarykey="emp_id"; // if want custom primary key in table

//If don't want then add : created_dt and update_dt

public $timestamps=false in user model class

Make Model

(Model name is capitalised and Below laravel 7 Model folder not available and user model also)

=>php artisan make:model User /

=>php artisan make:model User --migration // with migration file

=>php artisan make:model User -c --resource with controller with resource

MODEL / MIGRATION / CONTROLLER

ALL IN ONE : php artisan make:model User -c --resource --migration

Import Model in Controller

=>use App\Models\User; // add model   for use model functionality

Fetch Data by model in Mysql

```
Function fetchdata()
{
$data= User::all();  // fetch all data from users table
return view('viewusermodel',['data'=>$data]);  // load data in views
//return view('viewusermodel',compact('data'));  // load data in views convert in array

}
```

user.blade.php

```
@foreach ($data as $items)
<tr>
<td>{{$items->uid}}</td>
        <td>{{$items->unm}}</td>
        <td>{{$items->pass}}</td>
        <td>{{$items->email}}</td>
        <td>{{$items->mobile}}</td>

</tr>
@endforeach
```

Or

```
@foreach ($data as $items)
<tr>
<td>{{$items[‘uid’]}}</td>
        <td>{{$items[‘unm’]}}</td>
        <td>{{$items[‘pass’]}}</td>
        <td>{{$items[‘email’]}}</td>
        <td>{{$items[‘mobile’]}}</td>

</tr>
@endforeach

// if you want to add pagination
```

Pagination Fetch Data by model in Mysql

```
Function fetchdata()
{
$data= User::paginate(5);  // fetch all data from users table
return view('viewusermodel',['data'=>$data]);  // load data in views
}
```

Then just add below table list

```
<style>
.w-5{
        Display:none;
        width:30px;
    }
</style>
```

user.blade.php

```
@foreach ($data as $items)
<tr>
<td>{{$items->uid}}</td>
        <td>{{$items->unm}}</td>
        <td>{{$items->pass}}</td>
        <td>{{$items->email}}</td>
        <td>{{$items->mobile}}</td>

</tr>
@endforeach

<span >{{$data->links()}}</span>
```

Que: how can make custom in pagination view links

==========

18) Laravel Http Client : work with api & fetch data & display

What is http client

Introduction. Laravel provides an expressive, minimal API around the **Guzzle HTTP client**, allowing you to quickly make outgoing HTTP requests to communicate with other web applications. Laravel's wrapper around Guzzle is focused on its most common use cases and a wonderful developer experience.

Import Http Client in Controller

=>use Illuminate\Support\Facades\Http; // import Http client for fetch data from api

Fetch Data by model in Mysql

```
Function fetchdata()
{
$data=Http::get("http://reqres.in/api/users?page=1");  // fetch all data from api
return view('viewuserapi',['jsondata'=>$data['data']]);  // load data in view
}
```

user.blade.php

```
@foreach ($jsondata as $items)
<tr>
<td>{{$items['id']}}</td>
        <td>{{$items['email']}}</td>
        <td>{{$items['first_name']}}</td>
        <td>{{$items['last_name']}}</td>
        <td><img src={{$items['avatar']}}></td>
</tr>
@endforeach
```

==========

19) Laravel Http Request Method

The primary or most-commonly-used HTTP verbs (or methods, as they are properly called) are **POST, GET, PUT, PATCH, and DELETE**. These correspond to create, read, update, and delete (or CRUD) operations, respectively. There are a number of other verbs, too, but are utilized less frequently.

GET : When you fetching data (Default) // not use for securities purpose

If you use get method in form then use get method for get data

Route::get('getuser',[get_controller::class,'getmethod']); // by get method securities problem

POST : Posting or save data to server or database

If you use post method in form then use post method for posting data

Route::post('postuser',[post_controller::class,'postmethod']); // by post method

Note : also add @csrf in <form> tag for securities

<form method=”post”>

@csrf

</form>

PUT : When you update some data

Route::put('postuser',[post_controller::class,'postmethod']);

You **not use** direct put method like this **<form method=”PUT”>**

When you want to use put method then use **<form method=”POST”>**

& add

{{method_field(‘PUT’)}} IN FORM

PATCH : When update for specific field means (update where)

Route::patch('postuser',[post_controller::class,'postmethod']);

You **not use** direct put method like this **<form method=”PUT”>**

When you want to use put method then use **<form method=”POST”>**

& add

{{method_field(‘PATCH’)}} IN FORM

DELETE : When you want to delete some data

Route::delete('postuser',[post_controller::class,'postmethod']);

You **not use** direct put method like this **<form method=”PUT”>**

When you want to use put method then use **<form method=”POST”>**

& add

{{method_field(‘DELETE’)}} IN FORM

HEAD :add some extra parameters like Looking for headers

OPTIONS : Our Server are working or not

Que : What is diff between put & patch ?

==========

How to make a new Model/Controller/Migration in Laravel?

php artisan make:migration create_users_table // s

php artisan make:controller userController

php artisan make:model user

We create all bt this command

php artisan make:model user -m model with migration

php artisan make:model user -c -m Model/Controller/Migration

-a or — all Generate a migration, seeder, factory, and resource controller for the model

-c or — controller Create a new controller for the model

— force Create the class even if the model already exists

-m or — migration Create a new migration file for the model

Note :If you want to use custom table name or primary key or no auto_increement or no timestamp

Add variable in model class of table

Protected $table = “customers”;

Protected $primaryKey = “customer_id”;

Note:: it’s gives error because Laravel provide by default Column updated_at / created_at

If don’t want then add : **public $timestamps=false in user model class**

If don’t want then add : public $incrementing = false;

public $incrementing = false;

public $timestamps=false

==========

Crud In laravel

There are two to create crud in laravel

3. By Model / ORM
4. By Query Builder / DB CLASS

=> By Model

1. **View Data**

Import Model in Controller Note : table name users and model name user

=>use App\Models\User; // add model for use model functionality

Fetch Data by model in Mysql

Function fetchdata()

{

```
$data= User::all();  // fetch all data from users table
return view('viewusermodel',['data'=>$data]);  // load data in views
//return view('viewusermodel',compact('data'));  // load data by compact in views
}
```

ADD

user.blade.php

```
@foreach ($data as $items)
<tr>
<td>{{$items->uid}}</td>
        <td>{{$items->unm}}</td>
        <td>{{$items->pass}}</td>
        <td>{{$items->email}}</td>
        <td>{{$items->mobile}}</td>
</tr>
@endforeach
```

Or

```
@foreach ($data as $items)
<tr>
<td>{{$items['uid']}}</td>
        <td>{{$items['unm']}}</td>
        <td>{{$items['pass']}}</td>
        <td>{{$items['email']}}</td>
        <td>{{$items['mobile']}}</td>
</tr>
```

@endforeach

Save Data // make func in controller and get request from form

```
function add_user(Request $request)
	{
		//return $request->input(); // return all input type from form

		$Users= new User; // create obj of user model class

 $Users->email=$request->get('email');
				Or
		 $Users->email=$request['email'];
				or
		$Users->unm=$request->unm;
		$Users->pass=md5($request->pass);
		$Users->email=$request->email;
		$Users->mobile=$request->mobile;
		$Users->save(); // insert data in Users table
		Return redirect('/customer/view');
	}
```

Note:: it’s gives error because Laravel provide by default Column updated_at / created_at

If don’t want then add : **public $timestamps=false in user model class**

If don’t want then add : public $incrementing = false;

Custom name of timestamp

```
const CREATED_AT = 'creation_date';
const UPDATED_AT = 'updated_date';
```

Database connectivity

config/database.php/

```
 * The database connection that should be used by the model.
  protected $connection = 'sqlite';
```

Que : Can we save data in two table by one model

Delete data

user.blade.php

```
@foreach ($data as $items)
<tr>
<td>{{$items->id}}</td>
        <td>{{$items->unm}}</td>
        <td>{{$items->pass}}</td>
```

```
        <td>{{$items->email}}</td>
        <td>{{$items->mobile}}</td>
        <td><a href="{{url('customer/delete/')}}/{{$items->id}}">Delete</a></td>
</tr>
@endforeach
```

Delete Data

Note : when you use find() function it take primary key as default 'id'

If you want to set your own primary key then add in model class

```
protected $primaryKey = uid;
```

route.php

```
Route::get('customer/delete/{id}',[usersmodelcontroller::class,'delete']); // FETCH DATA FROM MODEL
```

view.php

```
<td><a href="{{url('customer/delete/')}}/{{$items->id}}">Delete</a></td>
```

Control.php

```
function delete($uid)
        {
                $Users=User::find($uid);
                if(!is_null($Users))
                {
                        $Users->delete();
```

```
}
                session()->flash('deleteuser','ses value'); // add flass session for msg in form
                return redirect('user');
        }
```

Update Data

Route.php

```
Route::get('customer/edit/{id}',[usersmodelcontroller::class,'edit_user']); // fetch data for edit
Route::post('update_user/id',[usersmodelcontroller::class,'update_user']); // update data
```

view.php

```
<td><a href="{{url('customer/edit/')}}/{{$d->id}}">Edit</a></td>
```

Control.php

```
function edit_user($uid)
        {
                $Users=User::find($uid);
                return view('edit_user',['data'=>$Users]); // fetch where data & load in view
        }

        function update_user(Request $req)
        {
                $Users=User::find($req->uid); // get uid for where update from hidden input
        $unm=$Users->unm=$req->unm;
                $Users->email=$req->email;
                $Users->mobile=$req->mobile;
```

```
                $Users->save(); // update data

                return redirect('user');
                //return redirect()->back();

        }
```

==========

Extra queries https://laravel.com/docs/8.x/eloquent

all data

//1) get all select data arr

$data = customer::all() return view('admin.manage_customer',['customer'=>$data]);

//2) get where condition data in arr get();

$arr=customer::where('cid','=','1')->where('sid','=','4')->get()

return view('admin.manage_customer',['customer'=>$arr]);

//3) where single data string format

$single=customer::where('email','=','raj@gmail.com')->first()

return view('admin.manage_customer',['customer'=>$single]);

//4) get single data by PK(ID)

$single=customer::find($id); // particular id data get in string

//5) paginate

$single=customer::all()->paginate(5);

$customer=customer::paginate(2); // select with paginate

$customer = customer::where('country', '=', 1)->paginate(15); // with where clause

display link in blade page

{{ $users->links() }} display the results and render the page links using Blade:

{{ $users->onEachSide(5)->links() }} Adjusting The Pagination Link Window

==========

select * from customer where id="1"

->customer::where('id','1')->get() // default operator =

Single where

->customer::where('id', '=', '1')->get() // get data in arr

Single where

->customer::where('id', '=', '1')->first() // get data in string

select * from customer where id=”1” and country=”india”

And Multiple

customer::where('id', '=', '1')->where('country', '=', 'india')->get();

Or

```
customer::where([
   ['column_name', 'operator', 'value'],
   ['another_column', 'operator', 'value']
])->get();
```

select * from customer where country=”india” or country=”japan”

Or Multiple

customer::where(‘country’,’=’,'india')->orWhere(‘country’,’=’,'japan')->get();

==========

Between

customer::whereBetween('age', [25,35])->get();

Join 2

```
$users = User::join('posts', 'users.id, '=','posts.user_id')->get();
```

```
$users = customer::join('countries', ‘customers.cid', '=','countries.id’)->get();
```

with as

```
$users = User::join('posts', 'users.id','=','posts.user_id')->get(['users.*',’posts.id as pid’]);
```

Join 3

```
$users = User::join('posts', 'posts.user_id', '=', 'users.id')
        ->join('comments', 'comments.post_id', '=', 'posts.id')->get()
```

Join with where

```
$users = User::join('posts', 'posts.user_id', '=', 'users.id')
      ->where('users.status',’=’,'active')
      ->where('posts.status',’=’,'active')->get();
```

==========

```
$result = customer::where('name'  ,  'LIKE',     '%'.$key.'%')
        ->orWhere('email',        'LIKE',        '%'.$key.'%')
        ->get();
```

```
$inquiries = Inquiry::orderBy('name', 'ASC')
  ->orderBy('created_at', 'DESC')
  ->get();
```

==========

Subquery Ordering

```
return Destination::addSelect(['last_flight' => Flight::select('name')
  ->whereColumn('destination_id', 'destinations.id')
  ->orderByDesc('arrived_at')
  ->limit(1)
])->get();
```

```
// Retrieve a model by its primary key...

$flight = customer::find($id);
```

Model Delete

```
$data=customer::find($id)->delete();
```

Model Update

```
$data=customer::find($id);
$data->name=$request->name;
$data->mobile=$request->mobile;
$data->update()/save()
```

==========

```
// Retrieve the first model matching the query constraints...
$flight = Flight::where('active', 1)->first();

// Alternative to retrieving the first model matching the query constraints...
$flight = Flight::firstWhere('active', 1);
```

Retrieving Aggregates

```
$count = customer::where('status', 'Unblock')->count();

$total = customer::all()->count('id');

$max = Flight::where('active', 1)->max('price');
```

==========

```
$students = Student::select("*")->whereBetween('points', [1, 150])->get();
```

==========

Call MYsql Procedure

```
DB::select('exec my_stored_procedure("Param1", "param2",..)');
```

==========

https://laravel.com/docs/11.x/queries#retrieving-a-list-of-column-values

=> By DB Class

First of all in control // import all DB class Query Builder

use Illuminate\Support\Facades\DB;

Retrieving All Rows From a Table

$users = DB::table('users')->get(); // select all

Retrieving a Single Row / Column From a Table

$user = DB::table('users')->where('name', 'John')->first();

retrieve a single row by its id column value, use the find method:

$user = DB::table('users')->find(3);

Retrieving a List of Column Values

$titles = DB::table('users')->pluck('title');

Aggregates

$users = DB::table('users')->count();

$price = DB::table('orders')->max('price');

Select Statements

$users = DB::table('users')

```
    ->select('name', 'email as user_email')
    ->get();

$users = DB::table('users')->distinct()->get();
```

Inner Join Clause

```
$users = DB::table('users')
    ->join('contacts', 'users.id', '=', 'contacts.user_id')
    ->join('orders', 'users.id', '=', 'orders.user_id')
    ->select('users.*', 'contacts.phone', 'orders.price')
    ->get();
```

Where Clauses

```
$users = DB::table('users')
        ->where('votes', '=', 100)
        ->where('age', '>', 35)
        ->get();
$users = DB::table('users')
        ->where('votes', '>=', 100)
        ->get();

$users = DB::table('users')
        ->where('votes', '<>', 100)
        ->get();

$users = DB::table('users')
        ->where('name', 'like', 'T%')
        ->get();
```

Or Where Clauses

```
$users = DB::table('users')
            ->where('votes', '>', 100)
            ->orWhere('name', 'John')
            ->get();


// fetch all data


function view_user()
        {
                $data= DB::table('users')->get();    // user::all()
                return view('view_user',['alldata'=>$data]);
        }
// fetch where data


function view_user()
        {
                $data= DB::table('users')->where('uid',10)->get();
return view('view_user',['alldata'=>$data]);
        }
```

Subquery Where Clauses

```
$users = User::where(function (Builder $query) {
  $query->select('type')
      ->from('membership')
      ->whereColumn('membership.user_id', 'users.id')
      ->orderByDesc('membership.start_date')
      ->limit(1);
}, 'Pro')->get();
```

Ordering

```
$users = DB::table('users')
            ->orderBy('name', 'desc')
            ->get();
$users = DB::table('users')
            ->orderBy('name', 'desc')
            ->orderBy('email', 'asc')
            ->get();
$user = DB::table('users')
            ->latest()
            ->first();
$randomUser = DB::table('users')
            ->inRandomOrder()
            ->first();
```

Insert Statements

```
DB::table('users')->insert([
  'email' => 'kayla@example.com',
  'votes' => 0
]);
```

Auto-Incrementing IDs

```
$id = DB::table('users')->insertGetId(
  ['email' => 'john@example.com', 'votes' => 0]
);
```

Upserts

The upsert method will insert records that do not exist and update the records that already exist with new values that you may specify. T

```
DB::table('flights')->upsert(
  [
    ['departure' => 'Oakland', 'destination' => 'San Diego', 'price' => 99],
    ['departure' => 'Chicago', 'destination' => 'New York', 'price' => 150]
  ],
  ['departure', 'destination'],
  ['price']
);
```

Update Statements

```
$affected = DB::table('users')
          ->where('id', 1)
          ->update(['votes' => 1]);
```

Update or Insert

```
DB::table('users')
  ->updateOrInsert(
    ['email' => 'john@example.com', 'name' => 'John'],
    ['votes' => '2']
  );
```

Delete Statements

```
$deleted = DB::table('users')->delete();

$deleted = DB::table('users')->where('votes', '>', 100)->delete();

DB::table('users')->truncate();
```

==========

Image Upload

**$file=**$request->file('img');

**$size=**$request->file('img')->getSize(); // want to get size

**$name=**$request->file('img')->getClientOriginalName(); // want to get name

**$extension=**$request->file('img')->getClientOriginalExtension(); // want to get Extension

If you want to make custom name then use below code

$filename= time() . '-fb.' . $request->file('img')->getClientOriginalExtension();

Echo $filename;   // output = 1625458754-ws.png

$request->file('img')->storeAs('uploads',$filename);

$data->img=$request->file('img')->store('uploads'); // store : storage/app/upload

$data->img=$request->file('img')->storeAs('uploads',$name); // store with original name

if($request->hasFile('img')**) check file or not**

{

unlink('upload/customer.'.$old_img);

}

else

{

}

Note : store in public folder

```
$file=$request->file('img');                $filename=time().'_img.'.$request->file('img')->getClientOriginalExtension();
$file->move('images/',$filename);  // use move for move image in public/images
$data->img=$filename; // name store in db
```

For view

```
<img src="{{ asset('images/' . $c->img)}}">
```

View image

```
<img src="{{ asset('storage/upload/'. $data->name) }}">
```

Multiple Image uploading

```
<input type="file" class="form-control" name="files[]" multiple />
```

validation

```
'images' => 'required',
'images.*' => 'mimes:jpg,png,jpeg,gif,svg'
```

```
$file=$request->file('files')
$filename=time().'_img.'.$file->extension();
$file->move('uplaod/customer',$filename);
```

```
$filesarr = [];
if($request->hasfile('files'))
 {
   foreach($request->file('files') as $file)
   {
      $name = time().rand(1000,9999).'_img.'.$file->extension();
      $file->move('uplaod/',$name);
      $filesarr[] = $name;
   }
     $data->file=implode(‘,’,$filesarr);
 }
```

//view multiple

```
$string_img=$fetch->img;
$arr_img=explode(',',$string_img);
@foreach($arr_img as $image)
        <div class="col-lg-4 col-sm-12 col-md-6 mb-3">
           <img src="uploads/{{$image}}" alt="{{$picture}}">
        </div>
@endforeach
```

// delete multiple

```
$data=customer::find($id)->first();
$string_img=$data->img;
@if($string_img!="")
$arr_img=explode(',',$string_img);
@foreach($arr_img as $image)
unlink('uploads/'.$image);
@endforeach
@enif
$data->delete();
```

==========

Search

View.blade.php

```
<form action="{{url{/manage_user}}}">
        <input type="search" name="search" value="{{$search}}">
<input type="submit" value="submit" name="submit">

<input type="submit" value="Reset" name="submit">
```

```
</form>
```

Usercontroller.php

```
Public function index(Request $request)
{
        $search=$request->search;
        if($search != “”)
{     // where  clause(‘col’, ‘=’ , ‘value’)
//$customer=customer::where(‘name’, ‘=’ ,$search)->get();
//$customer=customer::where(‘name’, ‘LIKE’ ,’%’.$search.’%’)->get(); // single col
$customer=customer::where(‘name’, ‘LIKE’ ,’%$search%’)->orWhere(‘email’, ‘LIKE’ ,’%’.$search.’%’)->get();  // multi col search

}
else
{
$customer=customer::all();
}
Return view(‘frontend.view’,compact(‘customer’,’search’));
}
```

web.php

```
Route::get('/manage_user',[customer_controller::class,'index']);
Route::post('/manage_user',[customer_controller::class,'index']);
```

==========

Install Sweet Alert Package

Staep 1: composer require realrashid/sweet-alert

Staep 2: php artisan sweetalert:publish

Setup Blade View

Staep 3: @include('sweetalert::alert')

Edit RegisterController

Staep 4: use RealRashid\SweetAlert\Facades\Alert;

or

use Alert;

Other Sweetalert Uses

```
Alert::success('Congrats', 'You\'ve Successfully Registered');
Alert::info('Info Title', 'Info Message');
Alert::warning('Warning Title', 'Warning Message');
Alert::error('Error Title', 'Error Message');
Alert::question('Question Title', 'Question Message')
Alert::image('Image Title!','Image Description','Image URL','Image Width','Image Height');
Alert::html('Html Title', 'Html Code', 'Type');
```

Use Helper Function

```
alert('Title','Lorem Lorem Lorem', 'success');
```

Copy

```
alert()->success('Title','Lorem Lorem Lorem');
```

Copy

```
alert()->info('Title','Lorem Lorem Lorem');
```

Copy

alert()->warning('Title','Lorem Lorem Lorem');

Copy

alert()->error('Title','Lorem Lorem Lorem');

Copy

alert()->question('Title','Lorem Lorem Lorem');

Copy

alert()->image('Image Title!','Image Description','Image URL','Image Width','Image Height');

Copy

alert()->html('HTML <u>example</u>'," You can use bold text, links and other HTML tags ",'success');

Toast

toast('Your Post as been submited!','success');

==========

Laravel 8 Authentication

Enter==> require net must

=>composer require laravel/ui /breeze // auth generate process

=>Php artisan ui vue - -auth

Or

composer require laravel/breeze

php artisan breeze/install

by this you can create by default laravel create Home controller page in auth folder

E:\xampp\htdocs\laravelapp\laraveltest\app\Http\Controllers\Auth

E:\xampp\htdocs\laravelapp\laraveltest\app\Http\resources\views/auth

Also login,regi,forgot,reset pass page

also create 2 migrate table file in for user_table & pass_table

E:\xampp\htdocs\laravelapp\laraveltest\database\migrations

Note : Now install npm for all dependencies for

Enter : npm install // its take some time as per net

Enter : npm run dev // gives error in laravel 8 so now add below code

Enter : npm run development

After That auth is completed & now you can see your login logout link on page

=> Now it's time for Database connectivity & migration for auth

Than Go CMD

Before migrate

than go E:\xampp\htdocs\laravelapp\config\database.php & remove mb4

Enter==> php artisan migrate // now check database

All table generate in Database

Note : If any Design issue then add BOOTSTRAP 5 SETUP

==========

Session :

Make Login with session

// Create session and store data

```
$request->session()->put('session_name',$data['userid']);
        session()->put('session_name',$value);
```

// Retrieving session

```
$request->session()->get('session_name')
    session()->get('session_name')
        session('session_name')
```

Exa:

```
echo session('session_name')
<h1>Hello : {{session('session_name')}}</h1>
```

// Retrieving all session

```
$request->session()->all()
        session()->all()

//Determining session available or not
if($request->session()->has('session_name'))
{

}
or
if(session('session_name'))
{

}
```

//Delete Session value for logout

```
$request->session()->pull('session_name');
        session()->pull('session_name');



//if sess then not open login page

Route::get('/login', function ()
{
    if(session()->has('session_name'))
    {
        return redirect('profile');
    }
```

```
        return view('login');
});

//if sess not then open login page

Route::get('/profile', function ()
{
        if(session()->has('session_name'))
        {
                return view('profile');
        }
        return redirect('login');
});

// logout
Route::get('/logout', function ()
{
        // delete session
        $request->session()->pull('session_name'); // single session delete
       $request->session()->forgot(['kes_name1',ses_name2]) // array session delete
        $request->session()->flush(); // all delete

        return redirect('login');
});
```

==========

Flash Session :

Flash messages is required in laravel application because that way we can give alter with what **progress complete**, error, warning etc. In this tutorial i added several way to give flash messages like redirect with success message, redirect with error message, redirect with warning message and redirect with info message.

Flash Session : destroy automatically after refresh one time

Exa: mail sent and session display then after refresh flash session destroy automatically means it's work one in while

Make Flash session

```
                                        //name  ,  value
$request->session()->flash('Success', 'Register Success'); // create flash session
```

```
return back()->with('Success', 'Register Success')
```

Use add page

```
@if(session('Success'))
<h3 style="color:green">{{session('Success')}} is added success</h3>
@endif
```

Que: How we use flash session 2 or 3 times after refresh

==========

Cookie in Laravel

1)Core method

```
setcookie('cookie_name','cookie_value',time()+15)  // set cookie
```

echo $*COOKIE[‘cookie*name’]; // get cookie

setcookie(‘cookie_name’,’cookie_value’,time()-15) // delete cookie

2)Using the Cookie facade:

use Illuminate\Support\Facades\Cookie;

// Create a new cookie and queue it for sending

Cookie::queue(Cookie::make('name', 'value', $minutes)); create

use Illuminate\Support\Facades\Cookie;

$value = Cookie::get('name'); // get

use Illuminate\Support\Facades\Cookie;

Cookie::queue(Cookie::forget('name')); // delete

3)Using the response helper function:

==========

Database Seeder and Faker in Laravel

Laravel includes the ability to seed your database with data using seed classes. All seed classes are stored in the database/seeders directory. By default, a DatabaseSeeder class is defined for you. From this class, you may use the call method to run other seed classes, allowing you to control the seeding order.

To generate a seeder, execute the make:seeder Artisan command. All seeders generated by the framework will be placed in the database/seeders directory:

A seeder class only contains one method by default: run. This method is called when the db:seed Artisan command is executed. Within the run method, you may insert data into your database however you wish. You may use the query builder to manually insert data or you may use Eloquent model factories.

Go : **php artisan make:seeder customerSeeder / create file in Database/seeder**

Go open file and load Model in seeder table file and create custom data query in run()

=>customerSeeder.php

```
$customer = new customer ;
$customer-> user_name='Rajesh';
$customer-> name='jesh';
$customer->email='raj@gmail.com';
$customer->gender='M';
$customer-> dob=now();
$customer->save();
```

Go open DatabaseSeeder.php / call function run()

```
$this->call([
        customerSeeder::class
])
```

Now Just fire command end entry done in database table

Go : php artisan db:seed

Faker : now want multiple different fake value in table

Go=>customerSeeder.php

```
Use Faker\Factory as faker;

 $faker= Faker::create();
      for ($i = 1; $i <= 50; $i++) {
         $data = new contact;
         $data->name = $faker->name; // $faker->sentence(3)
         $data->email = $faker->email;
         $data->mobile = $faker->numerify('##########');
         $data->comment = $faker->realText;;
         $data->save();
      }
```

Go : php artisan db:seed

==========

Custom Helper

we know laravel 11 also provides a helper function for array, url, route, path etc. But not all functions provided that we require.

Maybe some basic helper function like date format in our project. It requires a lot of time. so i think it's better we create our helper function and use the same code everywhere.

Step 1: Create app/helpers.php File

Step 2: Add File Path In composer.json File

```
"autoload": {

"files": [
    "app/helpers.php"
  ]
}
```

Step 3 : composer dump-autoload // reset all helper and add

Note: Now laravel include this page in all pages

Step 4: You can create your custom function in helpers.php

You can use as print data in <pre> tag

1.

```
if(!function_exists('p') ){

        Function p($data)
{
Echo "<pre>";
print_r($data);
```

Echo “</pre>”;

}

}

step 5: Now when you want data in <pre> just call anywhere in pages

p($request->all());

die;

2. For custom date format

if(!function_exists(‘custome_date’)){

Function custome_date($date,$format)

{

$date_formated=date($format,strtotime($date));

Return $date_formated;

}

}

Now when you want custom date then call

{{custome_date($d->dob, “d-M-Y”)}}

For image path setup

function productImagePath($image_name)

{

return public_path('images/products/'.$image_name);

}

==========

==========

Eloquent Relationship

Database tables are often related to one another. For example, a blog post may have many comments or an order could be related to the user who placed it. Eloquent makes managing and working with these relationships easy, and supports a variety of common relationships:

- One To One
- One To Many
- Many To Many
- Has One Through
- Has Many Through
- One To One (Polymorphic)
- One To Many (Polymorphic)
- Many To Many (Polymorphic)

One To One

A one-to-one relationship is a very basic type of database relationship. For example, a User model might be associated with one Phone model. To define this relationship, we will place a phone method on the User model. The phone method should call the hasOne method and return its result.
The hasOne method is available to your model via the model's Illuminate\Database\Eloquent\Model base class:

hasOne get one one type of data

Post_model.php make function

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

```
use Illuminate\Database\Eloquent\Relations\HasOne;
use Illuminate\Database\Eloquent\Factories\HasFactory;

class Post extends Model
{
    use HasFactory;

    /**
     * @return HasOne
     * @description get the detail associated with the post
     */
    public function detail(): HasOne
    {
        return $this->hasOne(Detail::class.'blog_id');
    }
}
```

details_model.php make function

```
use Illuminate\Database\Eloquent\Relations\BelongsTo;;
namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Illuminate\Database\Eloquent\Factories\HasFactory;

class Detail extends Model
{
```

```
    use HasFactory;

    /**
     * @return BelongsTo
     * @description Get the post that owns the details
     */
    public function post(): BelongsTo
    {
        return $this->belongsTo(Post::class);
    }
}
```

member_control.php

```
function index()
{
        //return Member::find(1)->post; // only group data fetch of id
        //return Member::with('post')->get(); // get all data from both table

}
```

One to Many

member_model.php make function

```
function getdata()
{
        return $this->hasMany('App\Models\group', 'group_id',''group_id'');
```

```
}

function index()
{
        //return Member::with('getdata')->get(); // get all data from both table

}
```

==========

Eloquent Mutator and Accessor

Accessors and mutators allow you to format Eloquent attributes when retrieving them from a model or setting their value. For example, you may want to use the Laravel encrypter to encrypt a value while it is stored in the database, and then automatically decrypt the attribute when you access it on an Eloquent model.

=> Eloquent Mutator : use for set attribute when you enter data in database

Go on model and create function / setNameAttribute($value)

```
Public function setNameAttribute($value)
{
        $this->attributes['name'] = ucwords($values)  // ucwords convert in capitalized
}
```

Note : If you want to add mutator o table column then use like

name setNameAttribute

user_name setUserNameAttribute

=> Eloquent Accessor : use for attribute when you get data from database

```
Public function getDobAttribute($value)
{
        Return date("d-M-Y",strtotime($value));
}
```

It shows all dob in view page as per above function

14) Laravel Form Validation

Use Validation Function

All Validation Rules comes from

Resource/lang/en/validation.php

Accepted

Accepted If

Active URL

After (Date)

After Or Equal (Date)

Alpha

Alpha Dash

Alpha Numeric

Array

Bail

Before (Date)

Before Or Equal (Date)

Between

Boolean

Confirmed

Current Password

Date

Date Equals

Date Format

Declined

Declined If

Different

Digits

Digits Between

Dimensions (Image Files)

Distinct

Email

Ends With

Enum

Exclude

Exclude If

Exclude Unless

Exclude Without

Exists (Database)

File

Filled

Greater Than

Greater Than Or Equal

Image (File)

In

In Array

Integer

IP Address

MAC Address

JSON

Less Than

Less Than Or Equal

Max

MIME Types

MIME Type By File Extension

Min

Multiple Of

Not In

Not Regex

Nullable

Numeric

Password

Present

Prohibited

Prohibited If

Prohibited Unless

Prohibits

Regular Expression

Required

Required If

Required Unless

Required With

Required With All

Required Without

Required Without All

Same

Size

Sometimes

Starts With

String

Timezone

Unique (Database)

URL

UUID

```
public function store(Request $request)
{
   $validated = $request->validate([
         'name' => 'required|alpha:ascii |max:255',
         'email' => 'required|unique:customers',
         'password' => 'required|min:8|max:12',
         'mobile' => 'required|digits:10',
         'gender'=> 'required|in:Male,Female',
         'hobby[]' => 'integer|boolean|min:0|max:2', 'required'
         'cid' => 'required',
         'img' => 'required|image|mimes:jpeg,png,jpg,gif,svg|max:2048'
      ]);
```

```
    // The blog post is valid...
}
```

Or

```
$validatedData = $request->validate([
    'title' => ['required', 'unique:posts', 'max:255'],
    'body' => ['required'],
]);
```

Customizing The Error Messages

Laravel's built-in validation rules each has an error message that is located in your application's resources/lang/en/validation.php file. Within this file, you will find a translation entry for each validation rule. You are free to change or modify these messages based on the needs of your application.

Stopping On First Validation Failure

Sometimes you may wish to stop running validation rules on an attribute after the first validation failure. To do so, assign the bail rule to the attribute:

```
$request->validate([
    'title' => 'bail|required|unique:posts|max:255',
    'body' => 'required',
]);
```

bail

Stop running validation rules for the field after the first validation failure.

While the bail rule will only stop validating a specific field when it encounters a validation failure, the stopOnFirstFailure method will inform the validator that it should stop validating all attributes once a single validation failure has occurred:

```
if ($validator->stopOnFirstFailure()->fails()) {
    // ...
}
```

A Note On Nested Attributes

If the incoming HTTP request contains "nested" field data, you may specify these fields in your validation rules using "dot" syntax:

```
$request->validate([
   'title' => 'required|unique:posts|max:255',
   'author.name' => 'required',
   'author.description' => 'required',
]);
```

Show all error Message

```
@if ($errors->any())
   <div class="alert alert-danger">
      <ul>
         @foreach ($errors->all() as $error)
            <li>{{ $error }}</li>
         @endforeach
      </ul>
   </div>
@endif
```

Show Single error by name Message

Now Show error use @error() @enderror

```
@error('name')
        <div class="alert alert-danger">{{ $message }}</div>
@enderror
```

Note : after error refresh page and all data reload then use {{old('name')}}

In value of input type

value="{{ old('name') }}"

For pass & conf pass use all time below **'password' and 'password_confirmation'**

'password' => 'required|confirmed',

'password_confirmation' => 'required',

Or custom wants then use **same:name**

'password' => 'required',

'password_conf' => 'required|same:password',

Manually Creating Validators

If you do not want to use the validate method on the request, you may create a validator instance manually using the Validator facade. The make method on the facade generates a new validator instance:

```
use Illuminate\Support\Facades\Validator;

public function store(Request $request)
  {
    $validator = Validator::make($request->all(), [
      'title' => 'required|unique:posts|max:255',
      'body' => 'required',
    ]);

    if ($validator->fails()) {
      return redirect('post/create')
            ->withErrors($validator)
            ->withInput();
```

```
    }
}
```

Error with every field

Interview Question

Why we use @csrf token

After submit and get error msg but how we get data in input type

15) Middleware

What is middleware

Middleware acts as a bridge between a request and a response. It is a type of filtering mechanism.

Most of use in Login as for authentication

Laravel includes a middleware that verifies whether the user of the application is authenticated or not. If the user is authenticated, it redirects to the home page otherwise, if not, it redirects to the login page.

Middleware type

We need to register each and every middleware before using it. There are Three types of Middleware in Laravel.

- Global Middleware: use and applied in complete website in single time
- Route Middleware: use and apply in specific single route at a time

- Group Middleware: use and applied in specific Route pages Exa if 20 page and want apply on 2 page // bunch of the routes

The Global Middleware will run on every HTTP request of the application, whereas the Route Middleware will be assigned to a specific route. The middleware can be registered at app/Http/Kernel.php. This file contains two properties: $middleware and $routeMiddleware. $middleware property is used to register Global Middleware and $routeMiddleware property is used to register route specific middleware.

Make Middleware

Middleware can be created by executing the following command −

php artisan make:middleware <middleware-name>

Middleware will be created at **app/Http/Middleware.** The newly created file will have the following code already created for you.

Register Middleware in **kernel.php**

1. Routes

```
 protected $routeMiddleware = [
      'user_after' => \App\Http\Middleware\user_after::class,      'user_before' => \App\Http\Middleware\user_before::class,
```

2. Group

```
 protected $middlewareGroups = [
       'admin_before'=>[ \App\Http\Middleware\admin_before::class],
       'admin_after'=> [\App\Http\Middleware\admin_after::class],
    ];
```

3. GLOBAL

```
protected $middleware = [
      // \App\Http\Middleware\TrustHosts::class,
   ];
```

Now : Laravel 11 No above kernel registered REQUIRED

https://stackoverflow.com/questions/78340907/how-do-i-register-a-middleware-in-laravel-11

==========

In Laravel 11, You can add/register middleware in bootstrap/app.php

```
$app->withMiddleware(function (Middleware $middleware) { $middleware->append(YourMiddleware::class); })
```

OR

```
->withMiddleware(function (Middleware $middleware) {
 $middleware->alias([ 'admin' => \App\Http\Middleware\AdminMiddleware::class, ]);
})
```

==========

Apply Global Middleware

For middleware use you have call middleware in **kernel.php in arr**

$middleware=[

\App\Http\Middleware\agecheck::class, // register it in var

Exa: Just create one global Middleware

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

                echo "<h1>Global middleware</h1>";

                // $request means provide url request or not
                // age request first present and then age less than 18

                // now just check in url ?age=10   like this

                if($request->age && $request->age<18)
                {
                        return redirect('noaccess');
                }
    return $next($request);
  }
```

==========

Group Middleware

- use and applied in specific pages or Routes EXa: if 20 page and want to apply on 10 page

Make Group Middleware & Register in kernel.php like Add in Route middleware Group in array

```
$middlewareGroups = [

        'groupmiddleware'=>[
            \App\Http\Middleware\groupmiddleware::class,
        ]
```

// as per Group middleware ?age=10 work only in particular belove Route not in above Route

Then call web.php like this

```
Route::group(['middleware'=>['groupmiddleware']],function(){

    Route::view('user','user');
    Route::view('about','about');

});
```

- Route Middleware : when we want to add middleware on specific route

Make RouteMiddleware & Register in kernel.php like Add in Route middleware in array

```
$routeMiddleware = [
        'routemiddleware'=>[
            \App\Http\Middleware\'routemiddleware'::class,
```

]

// as per Route middleware ?age=10 work only in one single Route

Then call web.php like this

Route::view('specificpage','specificpage')

Interview Que

Can we use two middleware in single Route

==========

Group Routes

Route Groups is an essential feature in Laravel, which allows you to group all the routes. Routes Groups are beneficial when you want to apply the attributes to all the routes. If you use route groups, you do not have to apply the attributes individually to each route; this avoids duplication. It allows you to share the attributes such as middleware or namespaces, without defining these attributes on each individual route. These shared attributes can be passed in an array format as the first parameter to the **Route::group** method.

1. Route::group([ ] , callback);

We use like this

Route::post('/manage_emp',[employee_controller::class,'store']);

Route::get('/manage_emp',[employee_controller::class,'index']);

Route::get('manage_emp/delete/{id}',[employee_controller::class,'destroy'])->name('manage_emp.delete');

```
Route::get('manage_emp/edit/{id}',[employee_controller::class,'edit']);
Route::post('manage_emp/update/{id}',[employee_controller::class,'update']);
```

Convert into Group Route

```
Route::group(['middleware'=>['groupmiddleware']],function(){

        Route::view('user','user');
        Route::view('about','about');

});
```

or

```
Route::group( [‘prefix’=>’/manage_emp’] , function(){
Route::post('/',[employee_controller::class,'store']);
Route::get('/',[employee_controller::class,'index']);
Route::get('/delete/{id}',[employee_controller::class,'destroy'])->name('manage_emp.delete');
Route::get('/edit/{id}',[employee_controller::class,'edit']);
Route::post(‘/update/{id}',[employee_controller::class,'update']);
});
```

Note : Remove all manage_customer set prefix on all group routes

==========

Email Sending

1. gmail login gmail

step : 1 setting=>all setting => Forwarding imap /pop

enable -> IMAP

enable -> pop

Save

Step : 2 google app => account => security

Enable : 2 step verification ON

step 3 : App => Account => search : App password =>

Generate app password =>

mail => windows computer => Generate password => copy pass & save in text

jfys wsuf inwi npsg

==========

Step 3 : open laravel env file / set as below

APP_NAME=Your Name

MAIL_MAILER=smtp

MAIL_HOST=smtp.gmail.com

MAIL_PORT=587

MAIL_USERNAME=vishvunjiya3058822@gmail.com // add email

MAIL_PASSWORD=hlddnycvcqludyef // add generate pass

MAIL_ENCRYPTION=tls

MAIL_FROM_ADDRESS=vishvunjiya3058822@gmail.com // add email

MAIL_FROM_NAME="${APP_NAME}"

Step : 4 Create MAIL

First Create Mail function in Laravel

=> php artisan make:mail welcomemail

it create file in App\Mail\welcomemail file

Step : 2

load below code in to controller.php

use illuminate\Support\Facades\Mail;

or

use Mail

=>**php artisan make:mail welcomemail**

app/mail/welcome

=> php artisan make:mail welcomemail -m Frontend.webisite.welcomeTemplate

also create with mail template

Method:2

=> go in .env

MAIL_MAILER=smtp

```
MAIL_HOST=smtp.googlemail.com
MAIL_PORT=587
MAIL_USERNAME=vishvunjiya3058822@gmail.com
MAIL_PASSWORD=fryanorgcpivbmjz
MAIL_ENCRYPTION=ssl
MAIL_FROM_ADDRESS=vishvunjiya3058822@gmail.com
MAIL_FROM_NAME="${APP_NAME}"
```

Cust_controller.php

```
use App\Mail\welcomemail;
use Mail;

$data=array(“name”=>$request->name)
Mail::to($email)->send(new welcomemail($data));
```

welcomemail.php

```
// pass $data in __cunstruct  and also pass variable which want to display in blade template page

public function __construct($data)
   {
      $this->data=$data;
   }

public function content()
   {
```

```
        return new Content(
            markdown: 'frontend.welcomeDes',
                                    with: [
                'msg' => $this->data['msg'],
                'sub' => $this->data['sub'],
            ],
        );
    }
```

welcomeDes.blade.php // use direct var in blade page

```
{{$msg}}
<br>
{{$sub}}
```

==========

SoftDeletes

// Use for temporary delete from table like / remove from cart / Move Thrash

Add below code in model page

Namespace: use Illuminate\Database\Eloquent\SoftDeletes;

Invoking: use SoftDeletes; // in class

Create again migrate file for add column (deleted_at) or add column first time when you create new migrate file

=> Php artisan make:migration add_deleted_at_to_customers_table

Then add some code in migration file

$table->SoftDeletes() in up()

$table->dropSoftDeletes() in down() function then migrate

=> Now you can see **deleted_at** column in customers table

=>Now after delete data from table / but record not delete from database and also shown deleted_at time in table list

Now want to retrieve data from table / functionality

withTrashed() // all data show Trashed and non Trashed

onlyTrashed() // data show all Trashed

restore() // rollback Trashed to original data

forceDelete() // delete from table so from trashed also

Delete() // temporary delete / so shown in trashed

// Show all data data

Function show_all_data()

{

$customer=customer::all();

Return view('data',['data'=>$customer])

```
}
Function trashed_data()
{
        $customer=customer::onlyTrashed()->get();
        Return view('data',['data'=>$customer])
}


// delete trash temporary


Function trash_delete($id)
{
        $customer=customer::find($id);
        $customer->delete()
}


Function restore($id)
{
        $customer=customer::withTrashed()->find($id);
        if(!is_null($customer))
{
$customer->restore();
}
}


// Permanent temporary


Function trash_delete($id)
{
```

```
        $customer=customer::withTrashed()->find($id);
        $customer->forceDelete();
Return back();
}
```

==========

Collective HTML Form package and Installation

Provide HTML form so we can replace html form from collective form

https://laravelcollective.com/docs/6.x/html

Go : composer require laravelcollective/html

After installation check | composer.json file / require laravelcollective/html load or not

Note: after Form::open (csrf) Not required

```
<h1>laravelcollective</h1>
{!! Form::open(['url' => 'foo/bar'])!!}</br><br>
{!! Form::text('username','Raj' ) !!}</br><br>
{!! Form::text('email', 'example@gmail.com')!!}</br><br>
{!! Form::password('password', ['class' => 'awesome','id' => 'awesome','placeholder' =>
'awesome'])!!}</br><br>
{!! Form::checkbox('name', 'value', true)!!}</br><br>
{!! Form::radio('name', 'value', true)!!}</br><br>
{!! Form::number('name', 'value')!!}</br><br>
{!! Form::date('name', \Carbon\Carbon::now())!!}</br><br>
{!! Form::file('image')!!}</br><br>
```

**{!! Form::select('size', ['L' => 'Large', 'S' => 'Small'], 'S')!!}</br>
**

{!! Form::close()!!}

Localization

Laravel's localization features provide **a convenient way to retrieve strings in various languages**, allowing you to easily support multiple languages within your application.

Website in multiple laug like japanese, english,endi,

Step: 1 : **resources/lang**

Create one file for multi language content **lang.php**

Create multi folder like

en

Hi

Ko

Gu

Then create similar file in all folder like **lang.php**

Add below code with array format with keywords

<?php

return ['welocme'=>'घर में आपका स्वागत है',

'home'=>'घर',

```
                'about'=>'के बारे में',
                'services'=>'सेवाएं',
                'blog'=>'ब्लॉग'
];

?>
```

Then call all keywords in main blade.php

```
<h1>@lang('lang.welcome')</h1>

<a href="#">@lang('lang.home')</a>
<a href="#">@lang('lang.about')</a>
<a href="#">@lang('lang.services')</a>
<a href="#">@lang('lang.blog')</a>
```

Step: 2 Define any language as Default

Go : config/app.php

Find 'locale'=> 'en' // Default laung set

Find 'fallback_cocale'=>'en' // not find any type then default take

Set any folder name ko , gu , hi

Step 3 : Now create multiple changes by dropdown or make link so set Routes

Web.php

```
Route::get('/{lang?}',function($lang==null){

        App::setlocale($lang);

        Return view('welcome');

})
```

==========

Laravel 8 Stub Customization

When we use Laravel artisan command to generate controller, model, factory, migration etc then we can see we get a code skeleton of each associated class or file. That well written code skeleton called as Laravel stub.

Laravel stub feature is included from laravel 7 version. Inside this article we will see Laravel 8 stub customization. If suppose we want to by default few methods, variables into class file of controller, model etc then we can generate.

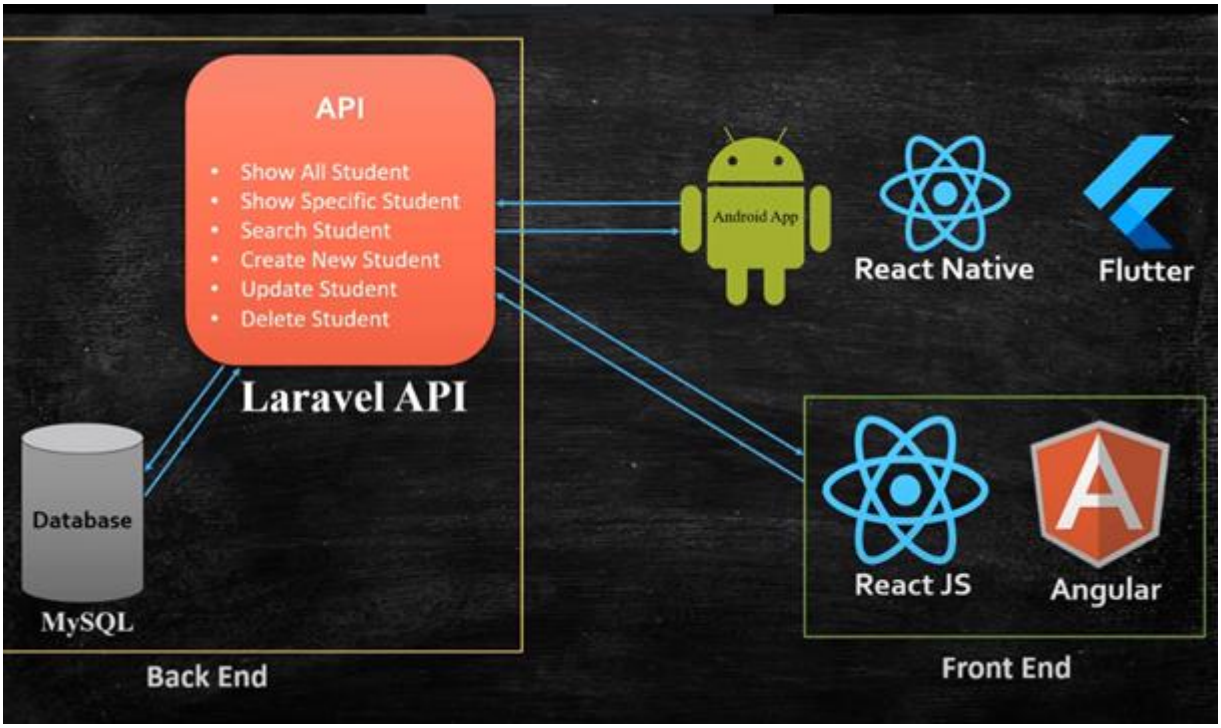
When we write any code in stub files then it will come with generated file. For example, if we add few methods in controller stub file, then whenever we generate controller file from php artisan command. Those added methods by default will come with file.

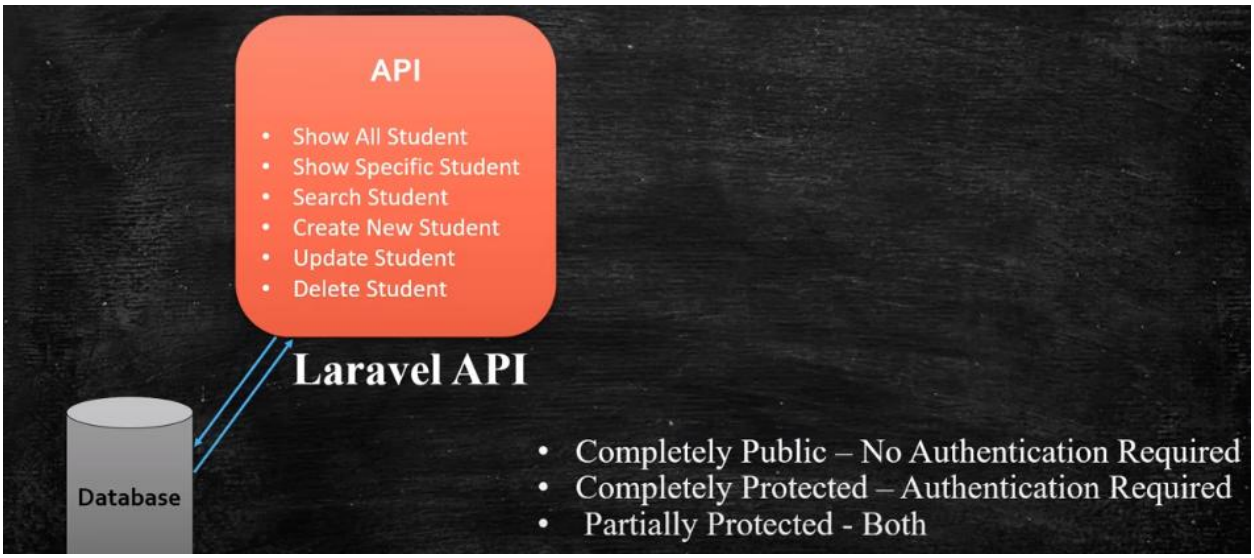
Make Stab : php artisan stub:publish

This command create the stubs folder at project root. When we open, we can see we have several files which is basically a skeleton file for migration, controller, model etc.

==========

Laravel API sanctum





Now For API we have to use Routes/api.php

Create Route in api.php

```
Route::get('/api_url',function(){
        return 'All api page';
})
```

- now for hit this route you have to add localhost/api/ api_url
- Note : if you want remove api/ from route

- GO : app/provider/RouteServiceProider.php
- remove : prefix(api) from boot() function

Now for web api check Download postman

www.postman.com => learning center /installing and updates / download any as per OS

Open it : Create New Collection => add new request=> then check all url in text box

Method:

Get : show data means get data

Post : Send data

==========

Now we go for project

So first create Model

cmd: php artisan make:model student -msc

Now Go in App/api.php

1. api.php : make Route
2. model : load in controller
3. Controller load in api.php

api.php Set all Routes

```
Route::get('/student',[student_controller::class,'allshow']); // datafetch
Route::post('/student',[student_controller::class,'store']); // insert
Route::post('/login',[student_controller::class,'student_login']); // login
Route::get('/student/{id}',[student_controller::class,'single_show']); // edit
Route::delete('/student/{id}',[student_controller::class,'destroy']); // delete
Route::put('/updatestudent/{id}',[student_controller::class,'update']); // update
Route::get('/search/{key}',[student_controller::class,'search']); // search
Route::put('/updatestatus/{id}',[student_controller::class,'updatestatus']); // block unblock
```

1) Now work on controller for api work

```
// Show all

public function allshow()
  {
    $data=student::all();
          return response()->json([
          'status'=>200,
          'students'=>$data
          ]);
  }

// Single_view / Edit

public function single_show($id)
  {
     $data=student::find($id);
                return response()->json([
                'status'=>200,
                'students'=>$data
                ]);
  }

// Search

function search($key)
  {
     $data=student::where('name','LIKE',"%$key%")->orWhere('email','LIKE','%'.$key.'%')->get();
                return response()->json([
                'status'=>200,
```

```
                'students'=>$data
                ]);
    }

// Insert



body // set

key value

public function store(Request $request)
    {

        $validate=Validator::make($request->all(),[
        'name'=>'Required',
        'email'=>'Required|email',
        'password'=>'Required'
    ]);

                if($validate->fails())
                {
                        return [
                                'success' => 0,
                                'message' => $validate->messages(),
                        ];
                }
                else
```

```
                {
                        $data=new student;
                        $data->name=$request->name;
                        $data->email=$request->email;
                        $data->password=Hash::make($request->password);

                        //create tocken
                        //$token=$data->createToken($data->email.'_Token')->plainTextToken;
                        $token=$data->token=Hash::make($request->email);
                        $data->save();

                        return response()->json([
                        'status'=>200,
                        'name'=>$data->name,
                        'token'=>$token,
                        'message'=>"Regioster Success"
                        ]);
                        //return            student::create($request->all());
                }
    }

// Update

public function update(Request $request, $id)
    {
        $validate=Validator::make($request->all(),[
            'name'=>'Required',
            'email'=>'Required|email',
            'password'=>'Required'
```

```
    ]);

                if($validate->fails())
                {
                        return [
                                'success' => 0,
                                'message' => $validate->messages(),
                        ];
                }
                else
                {
                        $data=student::find($id);
                        $data->name=$request->name;
                        $data->email=$request->email;
                        $data->password=$request->password;
                        $data->update();
                        return response()->json([
                        'status'=>200,
                        'message'=>"Update Success"
                        ]);
                }
  }

// Update status Block & Unblock

public function updatestatus(Request $request,$id)
  {
    $data=student::find($id);
                $status=$data->status;
```

```
                if($status === "Block")
                {
                        $data->status="Unblock";
                        $data->save();
                        return response()->json([
                        'status'=>200,
                        'msg'=>"Unblock Success"
                        ]);
                }
                else
                {
                        $data->status="Block";
                        $data->save();
                        return response()->json([
                        'status'=>200,
                        'msg'=>"Block Success"
                        ]);
                }
    }

// Delete

 public function destroy($id)
    {
        student::find($id)->delete();
                        return response()->json([
                        'status'=>200,
                        'msg'=>"Delete Success"
                        ]);
```

```
    }


// Login

public function student_login(Request $request)
        {
                $validate=Validator::make($request->all(),[
        'email'=>'Required|email',
        'password'=>'Required'
      ]);

                if($validate->fails())
                {
                        return [
                                'success' => 0,
                                'message' => $validate->messages(),
                        ];
                }
                else
                {
                        //$customer=student::where('email',$request->email)->first();
                        $student=student::where('email' , '=' , $request->email)->first();
                        if(! $student || ! Hash::check($request->password,$student->password))
                        {
                                return response()->json([
                                        'status'=>201,
                                        'msg'=>"student Login Failed due to Wrong Creadantial"
                                        ]);
```

```
                }
                else
                {

                        if($student->status=="Unblock")
                        {
                                return response()->json([
                                'status'=>200,
                                'msg'=>"student Login Success",
                                'name'=>$student->name,
                                'token'=>$student->token
                                ]);
                        }
                        else
                        {
                                return response()->json([
                                'status'=>201,
                                'msg'=>"student Blocked so Login Failed"
                                ]);
                        }
                }
        }
```

==========

Ajax In Laravel

php artisan route:cache

php artisan cache:clear

php artisan config:clear

Laravel Artisan Cache Commands Explained

Often times, when you are in the middle of developing a Laravel application, you may find that the changes you made in your code are not reflecting well on the application when testing.

Usually, the case is most likely caused by caching applied by the Laravel framework.

Here are some of the common commands you can run in your terminal to alleviate the issue.

❗ Make sure you are running them in the context of your application. Meaning, your terminal is currently in the same directory as your Laravel application.

1. Configuration Cache

Caching configuration helps with combining all of the configuration options for your application into a single file which will be loaded quickly by the framework.

Clearing Configuration Cache

However, if you notice changes to the configuration values in .env file is not reflecting on your application, you may want to consider clearing the configuration cache with the following command:

$ php artisan config:clear

Configuration cache cleared!

If you want to quickly reset your configuration cache after clearing them, you may instead run the following command:

$ php artisan config:cache

Configuration cache cleared!

Configuration cached successfully!

Caching your configuration will also help clear the current configuration cache. So it helps save your time without having to run both commands.

2. Route Caching

Caching your routes will drastically decrease the amount of time it takes to register all of your application's routes. When you add a new route, you will have to clear your route cache for the new route to take effect.

Clearing Route Cache

The following command will clear all route cache in your application:

$ php artisan route:clear

Route cache cleared!

To cache your routes again, simply run the following command:

$ php artisan route:cache

Route cache cleared!

Routes cached successfully!

Again, running the above command alone is enough to clear your previous route cache and rebuild a new one.

3. Views Caching

Views are cached into compiled views to increase performance when a request is made. By default, Laravel will determine if the uncompiled view has been modified more recently than the compiled view, before deciding if it should recompile the view.

Clearing View Cache

However, if for some reason your views are not reflecting recent changes, you may run the following command to clear all compiled views cache:

$ php artisan view:clear

Compiled views cleared!

In addition, Laravel also provides an Artisan command to precompile all of the views utilized by your application. Similarly, the command also clears the view cache before recompiling a new set of views:

$ php artisan view:cache

Compiled views cleared!

Blade templates cached successfully!

4. Events Cache

If you are using Events in your Laravel application, it is recommended to cache your Events, as you likely do not want the framework to scan all of your listeners on every request.

Clearing Events Cache

When you want to clear your cached Events, you may run the following Artisan command:

$ php artisan event:clear

Cached events cleared!

Likewise, caching your Events also clear any existing cache in the framework before a new cache is rebuilt:

$ php artisan event:cache

Cached events cleared!

Events cached successfully!

5. Application Cache

Using Laravel's Cache is a great way to speed up frequently accessed data in your application. While developing your application involving cache, it is important to know how to flush all cache correctly to test if your cache is working properly.

Clearing Application Cache

To clear your application cache, you may run the following Artisan command:

$ php artisan cache:clear

Application cache cleared!

This will clear all the cache data in storage which are typically stored in /storage/framework/cache/data/. The effect is similar to calling the Cache::flush(); Facade method via code.

❗ This command will NOT clear any config, route, or view cache, which are stored in /bootstrap/cache/ directory.

6. Clearing All Cache

Laravel provides a handy Artisan command that helps clear *ALL* the above caches that we have covered above. It is a convenient way to reset all cache in your application, without having to run multiple commands introduced before.

To clear all Laravel's cache, just run the following command:

$ php artisan optimize:clear

Compiled views cleared!

Application cache cleared!

Route cache cleared!

Configuration cache cleared!

Compiled services and packages files removed!

Caches cleared successfully!

As you can read from the terminal feedback, all cache types that existed in your Laravel application will be cleared entirely, except Events cache.

On live serve we don’t have cmd so you have add belove code in web.php

```
//Clear route cache
Route::get('/route-cache', function() {
    \Artisan::call('route:cache');
    return 'Routes cache cleared';
});

//Clear config cache
Route::get('/config-cache', function() {
    \Artisan::call('config:cache');
```

```
    return 'Config cache cleared';
});

// Clear application cache
Route::get('/clear-cache', function() {
    \Artisan::call('cache:clear');
    return 'Application cache cleared';
});

// Clear view cache
Route::get('/view-clear', function() {
    \Artisan::call('view:clear');
    return 'View cache cleared';
});

// Clear cache using reoptimized class
Route::get('/optimize-clear', function() {
    \Artisan::call('optimize:clear');
    return 'View cache cleared';
});
```

==========

Laravel Sanctum provides a featherweight authentication system for SPAs (single page applications), mobile applications, and simple, token based APIs. Sanctum allows each user of your application to generate multiple API tokens for their account. These tokens may be granted abilities / scopes which specify which actions the tokens are allowed to perform.

Installation

In new version all ready installed so check composer.json

if old version then add

composer require laravel/sanctum // load sanctum

php artisan vendor:publish --provider="Laravel\Sanctum\SanctumServiceProvider"// create migrate file

php artisan migrate

Next, if you plan to utilize Sanctum to authenticate an SPA, you should add Sanctum's middleware to your api middleware group within your application's app/Http/Kernel.php file:

```
'api' => [
  \Laravel\Sanctum\Http\Middleware\EnsureFrontendRequestsAreStateful::class,
  'throttle:api',
  \Illuminate\Routing\Middleware\SubstituteBindings::class,
],
```

Then add

To begin issuing tokens for users, your User model should use the Laravel\Sanctum\HasApiTokens trait:

```
use Laravel\Sanctum\HasApiTokens;

class User extends Authenticatable
{
  use HasApiTokens, HasFactory, Notifiable;
}
```

==========

// controller

6) make login & register by Laravel by default by (Auth)

This is scaffolding process

Enter==> php artisan ui vue --auth

cntroller page: path

by this you can create by default laravel create Home controller page in auth folder

E:\xampp\htdocs\laravelapp\laraveltest\app\Http\Controllers\Auth

E:\xampp\htdocs\laravelapp\laraveltest\app\Http\resources\views/auth

Also login,regi,forgot,reset pass page

also create 2 migrate table file in for user_table & pass_table

E:\xampp\htdocs\laravelapp\laraveltest\database\migrations

==========

7) Database connectivity

go xammp & create database laraveltest

than go E:\xampp\htdocs\laravelapp\laraveltest\.env file add this

DB_DATABASE=laraveltest

DB_USERNAME=root

DB_PASSWORD=

Now for Create automatecaly table which already store in

\laraveltest\database\migrations\2 file user & reset_pass table file

open both file & remove // unique() & index(); from file

Than Go CMD

Before migrate

than go E:\xampp\htdocs\laravelapp\config\database.php & remove mb4

Enter==> php artisan migrate // now check database

All table generate in Database

==========

Now final Command for dependency

Enter: npm install

Finally install npm than for finally authentication compilation to add remaining css and ect.

Enter: npm run dev

==========

8) custome table create with migration(file)

Enter==> php artisan make: migration custome_table

create migration file in \laraveltest\database\migrations\

than again\\Enter==> php artisan migrate // now check database

than go home & register & login & check

==========

9)also You can create Custome table

// this create full structure table file // so table keyword must in last

php artisan make:migration create_prod_table

// this create not structure table file

php artisan make:migration create_prod

after create table you can add field name

$table->string('title'); // than migrate

==========

10) Than make page & how make in router web

C:\xampp\htdocs\nagar\resources\views\wellcome.blade page call

//go in C:\xampp\htdocs\nagar\routes\web.php main for call all page from resources blade view

//go in resource / view/ copy home page & rename it but rename must .blade

make form add_prod

// also bootstrap func work like form-control,

//create one .blade.php file in resource/view

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

how to run

```
Route::view('/addprod', 'add_prod')
// call url func name,  page name

Route::post('/addprod', 'addController@store')//store function on this addcon file
```

==========

9) make model & controller for custome_table cured

Three type of create controller

1) php artisan make:controller prodController1

2) php artisan make:controller prodController2 --resource

3) php artisan make:controller prodController3 --resource --model=prod

than--- yes/no enter y //that create model& control both with all function

==========

Model: app/prod created

Add table in model // this model for only this table

```
{
        protected $table='prod';
        protected $primaryKey='id';
}
```

==========

controller: app/Http/Controller

In first method only controller page created

In Second method controller page created with funct index/create/store/show

In Third method controller page created with funct index/create/store/show & also autoload model page

==========

Blade component

https://www.youtube.com/watch?v=eSMV2JYffzo

https://www.youtube.com/watch?v=V86VhTsJ9Ww&list=PLjpp5kBQLNTQ-kR5LCIJpZ8nQGmw4mO5_&index=1

Theme Integration

Step 1 - Download Template

Step -2 Copy all css/js/img folder from template & copy in public folder

convert all page.blade.php

Step -3) Alaso make controller

If you want than create different controler

cmd=>php artisan make:controller TemplateControler

Than just create function like codeigniter function in control file

```
public function index()
	{
		return view('home');
	}

	public function shop()
	{
		return view('FrontEnd/shop');
	}
```

Step -4) than main route/web.php

```
Route::get('/', 'TemplateController@index');
Route::get('/home', 'TemplateController@index')->name('home');
Route::get('/shop', 'TemplateController@shop')->name('shop');
Route::get('/checkout', 'TemplateController@checkout')->name('checkout');
Route::get('/contact', 'TemplateController@contact')->name('contact');
Route::get('/product', 'TemplateController@shop')->name('product');
```