Laravel Fundamentals

Laravel Request Response :

Request -> Route -> Controller -> Return -> Response

Controller:

Controller files control the requests and responses

Controller file (DemoController.php) ->

```php
<?php

namespace App\Http\Controllers;

class DemoController extends Controller {

    public function demo1() {
        return response( 'Hello World!' );
    }
}
```

Route file (web.php) ->

```php
Route::get( 'demo1/', [DemoController::class, 'demo1'] );
```

Return types in controller methods:

```php
<?php

namespace App\Http\Controllers;

class DemoController extends Controller {

    public function demo1() {

        return response( 'Hello World!' );

    }

    // string
    public function demo2() {
        return response( 'This is a string!' );
    }
```

```
    // int
    public function demo3() {
        return response( 1 );
    }

    // null
    public function demo4() {
        return response( null );
    }

    // boolean
    public function demo5() {
        return response( true );
    }

    // array
    public function demo6() {
        return response( ['a', 'b', 'c'] );
    }

    // associative array or assoc
    public function demo7() {
        return response( [
            'name'    => 'Bill Gates',
            'company' => 'Microsoft',
        ] );
    }

    // multidimensional associative array
    public function demo8() {
        return response( [
            ['name1' => 'Bill Gates'],
            ['name2' => 'Microsoft'],
            ['name3' => 'Steve Jobs'],
        ] );
    }

    // binary large object (BLOB) -> files -> images, videos, etc; first save the
file in public folder then do the operation
    public function demo9() {
        return response()->file( 'image.png' );
    }

    // download response -> no file preview, direct download after hitting route
    public function demo10() {
        return response()->download( 'image.png' );
    }

    // redirect to another route
```

```
    public function demo11() {
        return redirect( 'demo2/' );
    }
}
```

```
<?php


use Illuminate\Support\Facades\Route;
use App\Http\Controllers\DemoController;

Route::get( 'demo1/', [DemoController::class, 'demo1'] );
Route::get( 'demo2/', [DemoController::class, 'demo2'] );
Route::get( 'demo3/', [DemoController::class, 'demo3'] );
Route::get( 'demo4/', [DemoController::class, 'demo4'] );
Route::get( 'demo5/', [DemoController::class, 'demo5'] );
Route::get( 'demo6/', [DemoController::class, 'demo6'] );
Route::get( 'demo7/', [DemoController::class, 'demo7'] );
Route::get( 'demo8/', [DemoController::class, 'demo8'] );
Route::get( 'demo9/', [DemoController::class, 'demo9'] );
Route::get( 'demo10/', [DemoController::class, 'demo10'] );
Route::get( 'demo11/', [DemoController::class, 'demo11'] );
```

Working with Laravel Request

Request Route Methods:

- GET
- POST
- PUT
- PATCH
- DELETE

```
<?php

namespace App\Http\Controllers;
use Illuminate\Http\Request;

class RequestResponseLearningController extends Controller {

    public function demo12( Request $request ) {
        $name = $request->name;
        $age  = $request->age;

        return response( ['name' => $name, 'age' => $age] );
    }
```

```
    public function demo13( Request $request ) {
        return response( $request->input() );
    }

    public function demo14( Request $request ) {
        return response( $request->input( "firstName" ) );
        // same:
        // return response( $request->firstName );
    }

    // with header
    public function demo15( Request $request ) {
        return response( $request->header() );
        // for specific header
        // return response( $request->header( "token" ) );
    }

    // query string
    public function demo16( Request $request ) {
        return response( $request->query() );
        // for specific
        // return response( $request->query( "name1" ) );
    }
}
```

```
Route::get( 'demo12/{name}/{age}/', [RequestResponseLearningController::class,
'demo12'] );
Route::get( 'demo13/', [RequestResponseLearningController::class, 'demo13'] );
Route::get( 'demo14/', [RequestResponseLearningController::class, 'demo14'] );
Route::get( 'demo15/', [RequestResponseLearningController::class, 'demo15'] );
Route::get( 'demo16/', [RequestResponseLearningController::class, 'demo16'] );
```

for demo12 url : http://127.0.0.1:8000/demo12/John/Doe
for demo16 url : http://127.0.0.1:8000/demo16?id1=1&id2=2&name1=John&name2=Doe

Working with multipart form data (Files with request):

working with file (getting the file, size of the file, original/temporary name of the file, extension):

```
public function demo20( Request $request ) {

        // picking a image file
        $photo = $request->file( 'image' );

        // size of the file
```

```
        $fileSize = $photo->getSize();

        // getting original name of the file
        $fileOriginalName = $photo->getClientOriginalName();

        // getting temporary name of the file
        $fileTemporaryName = $photo->getFilename();

        // getting file extension
        $fileExtensionWithExtension                  = $photo->extension();
        $fileExtensionWithGetClientOriginalExtension = $photo-
>getClientOriginalExtension();

        return response( [
            'fileSize'                                  => $fileSize,
            'fileOriginalName'                          => $fileOriginalName,
            'fileTemporaryName'                         => $fileTemporaryName,
            'fileExtensionWithExtension'                =>
$fileExtensionWithExtension,
            'fileExtensionWithGetClientOriginalExtension' =>
$fileExtensionWithGetClientOriginalExtension,
        ] );
    }
```

```
Route::post( 'demo20/', [WorkingWithMultipartformdataController::class, 'demo20']
);
```

Binary File Response:

```
public function demo29() {

        // getting the saved file path
        $filePath = "image.png";

        // to preview the file
        return response()->file( $filePath );
    }
```

```
Route::get( 'demo29/', [WorkingWithRedirectionController::class, 'demo29'] );
```

Binary File Download Response:

```
public function demo30() {

        // getting the saved file path
        $filePath = "image.png";
```

```
        // to download the file
        return response()->download( $filePath );
    }
```

```
Route::get( 'demo30/', [WorkingWithRedirectionController::class, 'demo30'] );
```

Moving the files to Storage/Public directory:

```
public function demo21( Request $request ) {

        // picking a image file
        $image = $request->file( 'image' );

        // to store the file in storage directory, with storeAs() method
        // storeAs(directory: 'uploads', save with the name: $image->getClientOriginalName() );
        $image->storeAs( 'uploads', $image->getClientOriginalName() );

        // to move the file in public directory, with move() method
        // move(directory: public_path(path: 'uploads' ), save with the name: $image->getClientOriginalName() );
        $image->move( public_path( 'uploads' ), $image->getClientOriginalName() );

        // keep that in mind, to save the file in both directory at once, first execute for storage and then for public

        return true;
    }
```

```
Route::post( 'demo21/', [WorkingWithMultipartformdataController::class, 'demo21'] );
```

Laravel IP address and Content:

ip address (ip address with request):

```
public function demo22( Request $request ) {

        // picking a image file
        $ipAddress = $request->ip();

        return response( $ipAddress );
    }
```

```
Route::post( 'demo22/', [WorkingWithMultipartformdataController::class, 'demo22']
);
```

Content Negotiations (Content Negotiations with request):

```
public function demo23( Request $request ) {

        // working with Content Negotiations
        // most of the time we use Accept header to pick the data format
        // there are many but mainly two types: text/html and application/json
        // mainly we work with application/json
        $acceptableContentTypes = $request->getAcceptableContentTypes();
        // will return an array with all the acceptable content types
        // for this case it will be [*/*], but we can even change this to
text/html or application/json etc, we can check on postman headers

        return response( $acceptableContentTypes );
    }
```

```
public function demo24( Request $request ) {
                // content type check
        if ( $request->accepts( 'application/json' ) ) {
            return 1; // true
        } else {
            return 0; // false
        }
    }
```

```
Route::post( 'demo23/', [WorkingWithMultipartformdataController::class, 'demo23']
);
Route::post( 'demo24/', [WorkingWithMultipartformdataController::class, 'demo24']
);
```

Working with Cookie (cookie with request):

```
// working with cookie
    public function demo25( Request $request ) {

        // getting all cookie value from request
        $cookieAll = $request->cookie();
        // getting specific cookie value from request
        $cookieSpecific = $request->cookie( 'Cookie_1' );

        return response( ['cookieAll' => $cookieAll,
```

```
            'cookieSpecific'            => $cookieSpecific] );
    }
```

```
Route::post( 'demo25/', [WorkingWithMultipartformdataController::class, 'demo25']
);
```

Returning Response in JSON format:

```
// working with json response
    public function demo26( Request $request ) {

        $array = [
            "firstName" => "John",
            "lastName"  => "Doe",
        ];

        return response()->json( $array, 201 ); // by default the status code
will be set to 200 which means 'ok', if we don't even pass any status code, but
if we pass any, then that will be set as the status code
    }
```

```
Route::post( 'demo26/', [WorkingWithMultipartformdataController::class, 'demo26']
);
```

Redirecting a Response to one route to another route:

```
public function demo27() {
        return redirect( 'demo28/' );
    }

    public function demo28() {
        return response( 'hello world!' );

    }
```

```
Route::get( 'demo27/', [WorkingWithRedirectionController::class, 'demo27'] );
Route::get( 'demo28/', [WorkingWithRedirectionController::class, 'demo28'] );
```

Setting Cookie:

```
public function demo31() {

        // generating and setting cookie
        $cookieName      = 'Cookie_1';
```

```
        $cookieValue     = 'Value_1';
        $cookieTimeInMin = 120;
        $path            = '/'; // will work on entire application
        $domain          = $_SERVER['SERVER_NAME']; // dynamically getting the
server name
        $secure          = false;
        $httpOnly        = true;

        return response( "hello" )->cookie( $cookieName, $cookieValue,
$cookieTimeInMin, $path, $domain, $secure, $httpOnly );
    }
```

```
Route::get( 'demo31/', [WorkingWithCookiesController::class, 'demo31'] );
```

Inserting Header:

```
public function demo32() {
        return response( "Hello" )
            ->header( 'key1', 'value1' )
            ->header( 'key2', 'value2' )
            ->header( 'key3', 'value3' );
    }
```

```
Route::get( 'demo32/', [RequestResponseLearningController::class, 'demo32'] );
```

View:

```
public function demo33() {

        // go to views folder -> create component folder -> create home.blade.php
        return view( 'component.home' );
    }
```

```
Route::get( 'demo33/', [ViewController::class, 'demo33'] );
```

Session:

```
// to put a data in session
    public function sessionPut( Request $request ) {
        $email = $request->email;
        $request->session()->put( 'userEmail', $email );

        return true;
```

```
    }

    // to pull session data
    public function sessionPull( Request $request ) {
        // with pull method, session data can be retrieved only one time, then
that will be flushed
        $sessionPull = $request->session()->pull( 'userEmail', 'defualt' );

        return response( $sessionPull );
    }

    // to get session data
    public function sessionGet( Request $request ) {
        // with get method, session data can be retrieved many times, won't be
flushed
        $sessionGet = $request->session()->get( 'userEmail', 'defualt' );

        return response( $sessionGet );
    }

    // to forget a specific session data
    public function sessionForget( Request $request ) {
        $request->session()->forget( 'userEmail' );

        return true;
    }

    // to flush all session data
    public function sessionFlush( Request $request ) {
        $request->session()->flush();

        return true;
    }
```

```
Route::get( 'demo35/{email}', [SessionController::class, 'sessionPut'] );
Route::get( 'demo36/', [SessionController::class, 'sessionPull'] );
Route::get( 'demo37/', [SessionController::class, 'sessionGet'] );
Route::get( 'demo38/', [SessionController::class, 'sessionForget'] );
Route::get( 'demo39/', [SessionController::class, 'sessionFlush'] );
```

Another :

```
    public function setSession( Request $request )
    {
        session()->put( 'name', 'john doe' );
        $request->session()->put( 'email', 'johndoe@gmail.com' );
        session( ['phone' => '0123456789'] );
```

```
        return response( 'session set' );
    }

    public function getSession( Request $request )
    {
        $name  = session()->get( 'name' );
        $email = $request->session()->get( 'email' );
        $phone = session( 'phone' );
        $age   = session( 'age', 25 );

        return response( compact( 'name', 'email', 'phone', 'age' ) );
    }

    public function readInView( Request $request )

    {
        $name  = session()->get( 'name' );
        $email = $request->session()->get( 'email' );
        $phone = session( 'phone' );
        $age   = session( 'age', 25 );

        return view( 'about', compact( 'name', 'email', 'phone', 'age' ) );
    }

    public function updateSession( Request $request )
    {
        session()->put( 'name', 'jane doe' );
        $request->session()->put( 'email', 'janedoe@gmail.com' );
        session( [
            'phone' => '0123456788',
            'age'   => 30,
        ] );

        return response( 'session updated' );
    }

    public function destroySession( Request $request )
    {
        session()->forget( 'name' );
        $request->session()->forget( 'email' );
        session()->forget( [
            'phone',
            'age',
        ] );

        return response( 'session destroyed' );
    }
```

```
Route::get( 'demo/', [DemoController::class, 'setSession'] );
Route::get( 'demo2/', [DemoController::class, 'getSession'] );
Route::get( 'demo4/', [DemoController::class, 'readInView'] );
Route::get( 'demo5/', [DemoController::class, 'updateSession'] );
Route::get( 'demo6/', [DemoController::class, 'destroySession'] );
```

Session With Flash (Will execute only one time after hitting the route):

```
    // for this, only one time use available, after hitting the route, for first time, the flash message will be shown, for next time, that will be gone
    public function setSessionFLashMessage( Request $request )
    {
        session()->flash( 'message', 'a flash message' );
        $request->session()->flash( 'error', 'an error flash message' );
        session( ['phone' => '0123456789'] );

        return response( 'flash message set' );
    }

    public function getSessionFlashMessage( Request $request )
    {
        $message = session()->get( 'message' );
        $error   = $request->session()->get( 'error' );

        return response( compact( 'message', 'error' ) );
    }

    public function flash()
    {
        return view( 'flash' );
    }
```

flash.blade.php -

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Read Flash</title>
</head>

<body>
    <h1>Read Flash</h1>
```

```
    @if (session('message'))
        <p>{{ session('message') }}</p>
    @else
        <p>No message</p>
    @endif
</body>

</html>
```

```
Route::get( 'demo7/', [DemoController::class, 'setSessionFLashMessage'] );
Route::get( 'demo8/', [DemoController::class, 'getSessionFlashMessage'] );
Route::get( 'demo9/', [DemoController::class, 'flash'] );
```

Logging:

```
public function demo34( Request $request ) {

        $num1 = $request->num1;
        $num2 = $request->num2;
        $sum  = $num1 + $num2;

        Log::info( $sum );
        Log::error( $sum );
        Log::warning( $sum );

        return response( $sum );
    }
```

```
Route::get( 'demo34/{num1}/{num2}', [LogController::class, 'demo34'] );
```

Middleware:

How we define middleware:

- First we create middleware
- Define the logics in that middleware
- Create alias of that middleware class, or give a name of that middleware class, in bootstrap folder -> app.php file
- Then we call that middleware by that alias name, with the routes at the end, or we can even create groups

php artisan make:middleware 'miidlewareName'

php artisan make:middleware DemoMiddleware

with Header property check:

```php
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;
use Symfony\Component\HttpFoundation\Response;

class DemoMiddleware {
    /**
     * Handle an incoming request.
     *
     * @param  \Closure(\Illuminate\Http\Request): (\Symfony\Component\HttpFoundation\Response)  $next
     */
    public function handle( Request $request, Closure $next ): Response {
        $key = $request->header( 'apiKey' );
// condition to check
        if ( $key == 1234 ) {
            return $next( $request );
        } else {
            return response()->json( 'unauthorized', 401 );
        }
    }
}
```

```php
<?php

namespace App\Http\Controllers;

class MiddlewareController extends Controller {

    public function demo40() {
        return "hello";
    }
}
```

```php
Route::get( 'demo40/', [MiddlewareController::class, 'demo40'] )->middleware( [DemoMiddleware::class] );
```

Redirection with middleware:

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

```
        $key = $request->key;

// condition to check
        if ( $key == 1234 ) {
            return $next( $request );
        } else {
            return redirect( 'demo41/' );
        }
    }
```

```
public function demo40() {

        return "hello1";
    }

    public function demo41() {

        return "hello2";
    }
```

```
Route::get( 'demo40/{key}', [MiddlewareController::class, 'demo40'] )-
>middleware( [DemoMiddleware::class] );
Route::get( 'demo41/', [MiddlewareController::class, 'demo41'] );
```

Middleware Groups:

Directly declaring the class:

```
// directly declaring the class
Route::middleware( [DemoMiddleware::class] )->group( function () {
    Route::get( 'demo41/{key}', [MiddlewareController::class, 'demo41'] );
    Route::get( 'demo42/{key}', [MiddlewareController::class, 'demo42'] );
    Route::get( 'demo43/{key}', [MiddlewareController::class, 'demo43'] );
    Route::get( 'demo44/{key}', [MiddlewareController::class, 'demo44'] );
} );
```

Using the alias name, after declared the class in the app.php

```
// using the alias name, after declared the class in the app.php
Route::middleware( ['demoMiddleware'] )->group( function () {
    Route::get( 'demo41/{key}', [MiddlewareController::class, 'demo41'] );
    Route::get( 'demo42/{key}', [MiddlewareController::class, 'demo42'] );
    Route::get( 'demo43/{key}', [MiddlewareController::class, 'demo43'] );
    Route::get( 'demo44/{key}', [MiddlewareController::class, 'demo44'] );
} );
```

With alias(),

```php
<?php

use Illuminate\Foundation\Application;
use App\Http\Middleware\DemoMiddleware;
use Illuminate\Foundation\Configuration\Exceptions;
use Illuminate\Foundation\Configuration\Middleware;

return Application::configure( basePath: dirname( __DIR__ ) )
    ->withRouting(
        web: __DIR__ . '/../routes/web.php',
        commands: __DIR__ . '/../routes/console.php',
        health: '/up',
    )
    ->withMiddleware( function ( Middleware $middleware ) {
        $middleware->alias( [
            'demoMiddleware' => DemoMiddleware::class,
        ] );
    } )
    ->withExceptions( function ( Exceptions $exceptions ) {
        //
    } )->create();
```

to apply middleware in all routes declared in web.php or api.php (go to bootstrap file -> app.php file):

```php
<?php

use Illuminate\Foundation\Application;
use App\Http\Middleware\DemoMiddleware;
use Illuminate\Foundation\Configuration\Exceptions;
use Illuminate\Foundation\Configuration\Middleware;

return Application::configure( basePath: dirname( __DIR__ ) )
    ->withRouting(
        web: __DIR__ . '/../routes/web.php',
        commands: __DIR__ . '/../routes/console.php',
        health: '/up',
    )
    ->withMiddleware( function ( Middleware $middleware ) {
        $middleware->web( append: [
            'demoMiddleware' => DemoMiddleware::class,
        ] );
        $middleware->api( prepend: [
             'demoMiddleware' => DemoMiddleware::class,
         ] );
    } )
```

```
    ->withExceptions( function ( Exceptions $exceptions ) {
        //
    } )->create();
```

Manipulate Headers in Middleware:

Adding Header in middleware:

```
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;
use Symfony\Component\HttpFoundation\Response;

class DemoMiddleware {
    /**
     * Handle an incoming request.
     *
     * @param  \Closure(\Illuminate\Http\Request): (\Symfony\Component\HttpFoundation\Response)  $next
     */
    public function handle( Request $request, Closure $next ): Response {
        //adding header from middleware, after hitting the route, it will come to middleware and this header will be added
        $request->headers->add( ['email' => 'john@gmail.com'] );
        return $next( $request );
    }
}
```

```
public function demo45( Request $request ) {
        return $request->header();
    }
```

```
Route::get( 'demo45', [MiddlewareController::class, 'demo45'] )->middleware( [DemoMiddleware::class] );
```

Replacing Header in middleware:

```
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;
```

```
use Symfony\Component\HttpFoundation\Response;

class DemoMiddleware {
    /**
     * Handle an incoming request.
     *
     * @param  \Closure(\Illuminate\Http\Request):
(\Symfony\Component\HttpFoundation\Response)  $next
     */
    public function handle( Request $request, Closure $next ): Response {
        //Replacing header from middleware, after hitting the route, it will come
to middleware and this header will be replaced, to that previously set header
        $request->headers->replace( ['email' => 'john@gmail.com'] );
        return $next( $request );
    }
}
```

```
public function demo45( Request $request ) {
        return $request->header();
    }
```

```
Route::get( 'demo45', [MiddlewareController::class, 'demo45'] )->middleware(
[DemoMiddleware::class] );
```

Removing Header in middleware:

```
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;
use Symfony\Component\HttpFoundation\Response;

class DemoMiddleware {
    /**
     * Handle an incoming request.
     *
     * @param  \Closure(\Illuminate\Http\Request):
(\Symfony\Component\HttpFoundation\Response)  $next
     */
    public function handle( Request $request, Closure $next ): Response {
        //Removing header from middleware, after hitting the route, it will come
to middleware and this previously set header will be removed
        $request->headers->remove( 'email' ); // for removing, we need to only
set the key
        return $next( $request );
```

```
    }
}
```

```
public function demo45( Request $request ) {
        return $request->header();
    }
```

```
Route::get( 'demo45', [MiddlewareController::class, 'demo45'] )->middleware(
[DemoMiddleware::class] );
```

Throttle: Request rate limiting in middleware (for one route):

```
public function demo46() {
        return "hello";
    }
```

```
// This was for working with only one route
Route::get( 'demo46', [MiddlewareController::class, 'demo46'] )->middleware(
'throttle:5,1' );
```

Example of group middleware with throttle:

```
Route::middleware( 'throttle:5,1' )->group( function ()
{
    Route::get( 'demo/', [DemoController::class, 'setSession'] );
    Route::get( 'demo2/', [DemoController::class, 'getSession'] );
    Route::get( 'demo3/', [DemoController::class, 'readAgain'] );
    Route::get( 'demo4/', [DemoController::class, 'readInView'] );
    Route::get( 'demo5/', [DemoController::class, 'updateSession'] );
    Route::get( 'demo6/', [DemoController::class, 'destroySession'] );

    Route::get( 'demo7/', [DemoController::class, 'setSessionFLashMessage'] );
    Route::get( 'demo8/', [DemoController::class, 'getSessionFlashMessage'] );
    Route::get( 'demo9/', [DemoController::class, 'flash'] );

    Route::get( '/viewForm', [FormController::class, 'viewForm'] );
    Route::post( '/submitForm', [FormController::class, 'submitForm'] )->name(
'submitForm' );
} );
```

Request rate limiting in middleware (for entire application, web.php):

```
$middleware->web( append: [
            'throttle:10,1',
```

```
        ] );
```

Constructor in Controller:

As we all know, constructors executed by themselves automatically at the beginning, no need to call particularly. We can define middleware through these constructors. Like - if we want that a middleware will work only in a specific controller, then for that controller we can add that middleware, no need to add middleware with the routes particularly.

```

```

```

```

Controller Types:

- **Basic Controller**

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class BasicController extends Controller {

    public function demo47( Request $request ) {
        $name = $request->name;

        return response( $name );
    }
}
```

```
Route::get( 'demo47/{name}', [BasicController::class, 'demo47'] );
```

- **Single Action Controller** : There will be only one function and it's executed by a method called _invoke()__ :
- php artisan make:controller SingleActionController --invokable

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
```

```
class BasicController extends Controller {

    public function demo47( Request $request ) {
        $name = $request->name;

        return response( $name );
    }
}
```

In route, no need to take an array [] to pass the controller class and the method. Here for this, just we need to mention the class.

```
Route::get( 'demo47/{name}', SingleActionController::class );
```

- **Resource Controller** : There will be generated all the methods, which we need for CRUD operations
- php artisan make:controller ResourceController --resource

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class ResourceController extends Controller
{
    /**
     * Display a listing of the resource.
     */
    public function index()
    {
        //
    }

    /**

     * Show the form for creating a new resource.

     */
    public function create()
    {
        //
    }

    /**
     * Store a newly created resource in storage.
     */
```

```
    public function store( Request $request )
    {
        //
    }

    /**
     * Display the specified resource.
     */
    public function show( string $id )
    {
        //
    }

    /**
     * Show the form for editing the specified resource.

     */
    public function edit( string $id )
    {
        //
    }

    /**
     * Update the specified resource in storage.
     */
    public function update( Request $request, string $id )
    {
        //
    }

    /**
     * Remove the specified resource from storage.
     */
    public function destroy( string $id )
    {
        //
    }
}
```

In route, no need to take an array [] to pass the controller class and the method. Here for this, just we need to mention the class and the basic endpoint, rest resource method will generate automatically. Keep that in mind, methods generated in ResourceController should not be changed at any cost, otherwise routes won't work.

```
Route::resource( 'demo49/', ResourceController::class );
```

Route Lists for that ResourceController:

```
GET|HEAD        demo49 ..................................... index › ResourceController@index
  POST          demo49 ..................................... store › ResourceController@store
  GET|HEAD      demo49/create .............................. create › ResourceController@create
  GET|HEAD      demo49/{} .................................. show › ResourceController@show
  PUT|PATCH     demo49/{} .................................. update › ResourceController@update
  DELETE        demo49/{} .................................. destroy › ResourceController@destroy
  GET|HEAD      demo49/{}/edit ............................. edit › ResourceController@edit
```

Blade:

```php
<?php

namespace App\Http\Controllers;

class BladeController extends Controller
{
    public function demo51()
    {
        // multidimensional associative array
        $billGates = [
            ["firstName" => "Bill", "lastName" => "Gates", "age" => 68],
            ["firstName" => "Mark", "lastName" => "Zuckerberg", "age" => 62],
            ["firstName" => "Jeff", "lastName" => "Bezos", "age" => 57],
        ];

        return view( 'demo51', compact( 'billGates' ) );
        // can also write without compact() function:
        // return view( 'demo51', ['billGates' => $billGates] );
    }
}
```

in demo51.blade.php in views folder:

```
<body>

    <h1>This is demo 51</h1>

    <ol>
        @foreach ($billGates as $person)
            <li>
```

```
                User name is = {{ $person['firstName'] }} and age is = {{
$person['age'] }}
            </li>
        @endforeach
    </ol>

</body>
```

```
Route::get( 'demo51/', [BladeController::class, 'demo51'] );
```

Validation:

```
public function viewForm()
    {
        return view( 'form' );
    }

    public function submitForm( Request $request )
    {
        $request->validate( [
            'name'                  => 'required|min:3',
            'email'                 => 'required|email',
            'password'              => 'required|min:4',
            'password_confirmation' => 'required|same:password',
        ] );

        return response( 'form submitted successfully' );
    }
```

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Form</title>
</head>

<body>
    <h1>Form</h1>

    <form action="{{ route('submitForm') }}" method="POST">
        @csrf

        <label for="name">Name:</label>
        <input type="text" name="name" id="name" placeholder="Name" value="{{
```

```
old('name') }}">
        @error('name')
            <p>{{ $message }}</p>
        @enderror


        <label for="email">Email:</label>
        <input type="email" name="email" id="email" placeholder="Email" value="{{
old('email') }}">
        @error('email')
            <p>{{ $message }}</p>
        @enderror


        <label for="password">Password:</label>
        <input type="password" name="password" id="password"
placeholder="Password" value="{{ old('password') }}">
        @error('password')
            <p>{{ $message }}</p>
        @enderror


        <label for="password_confirmation"></label>
        <input type="password" name="password_confirmation"
id="password_confirmation"
            placeholder="Password Confirmation" value="{{
old('password_confirmation') }}">
        @error('password_confirmation')
            <p>{{ $message }}</p>
        @enderror

        <button type="submit">Submit</button>
    </form>
</body>

</html>
```

```
Route::get( '/viewForm', [FormController::class, 'viewForm'] );
Route::post( '/submitForm', [FormController::class, 'submitForm'] )->name(
'submitForm' );
```

Migration:

Commands:

Create a new table:

php artisan make:migration create_tableName

Add a new column:
php artisan make:migration add_column_to_tableName

Rename column:
php artisan make:migration rename_column_to_tableName

Update a column like rename column:
php artisan make:migration update_column_to_tableName

Remove a column:
php artisan make:migration remove_column_from_tableName

Drop a column:
php artisan make:migration drop_column_from_profiles

Drop a table:
php artisan make:migration drop_tableName

Drop all tables:
php artisan migrate:reset

Available Column Types:
https://laravel.com/docs/11.x/migrations#available-column-types

Available Column Modifiers/Attributes:
https://laravel.com/docs/11.x/migrations#column-modifiers

Available Relationship Contraints:
https://laravel.com/docs/11.x/migrations#foreign-key-constraints

Table Creation:

```
php artisan make:migration create_profiles
```

```php
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
```

```
        Schema::create( 'profiles', function ( Blueprint $table )
        {
            $table->bigIncrements( 'id' );
            $table->boolean( 'isBangladeshi' );
            $table->bigInteger( 'votes' );
            $table->mediumInteger( 'medium' );
            $table->smallInteger( 'small' );
            $table->tinyInteger( 'tiny' );
            $table->decimal( 'decimal', 5, 2 );
            $table->binary( 'image' );
            $table->char( 'name', 50 );
            $table->dateTime( 'votingDateTime' );
            $table->time( 'votingTime' );
            $table->timestamp( 'votingTimestamp' );
            $table->date( 'votingDate' );
            $table->double( 'people' );
            $table->enum( 'group', ['male', 'female'] );
            $table->float( 'dollar' );
            $table->geometry( 'positions' );
            $table->ipAddress( 'ipAddress' );
            $table->integer( 'age' );
            $table->json( 'data' );
            $table->longText( 'description' );
            $table->mediumText( 'text' );
            $table->unsignedBigInteger( 'user_id' );
            $table->unsignedInteger( 'post_id' );
            $table->unsignedMediumInteger( 'medium_int' );
            $table->unsignedSmallInteger( 'small_int' );
            $table->unsignedTinyInteger( 'tiny_int' );
            $table->text( 'body' );
            $table->tinyText( 'title' );
            $table->timestamps();
        } );
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists( 'profiles' );
    }
};
```

migration command:

```
php artisan migrate
```

More:

```
            $table->bigIncrements( 'id' );

            $table->string( 'email', 50 )->unique();
            $table->string( 'firstName', 50 );
            $table->string( 'lastName', 50 )->nullable();
            $table->string( 'country', 50 )->default( 'Bangladesh' );

            $table->boolean( 'isBangladeshi' );
            $table->bigInteger( 'votes' );
            $table->mediumInteger( 'medium' );
            $table->smallInteger( 'small' );
            $table->tinyInteger( 'tiny' );
            $table->decimal( 'decimal', 5, 2 );
            $table->binary( 'image' );
            $table->dateTime( 'votingDateTime' );
            $table->time( 'votingTime' );
            $table->timestamp( 'votingTimestamp' );
            $table->date( 'votingDate' );
            $table->double( 'people' );
            $table->enum( 'group', ['male', 'female'] );
            $table->float( 'dollar' );
            $table->geometry( 'positions' );
            $table->ipAddress( 'ipAddress' );
            $table->integer( 'age' );
            $table->json( 'data' );
            $table->longText( 'description' );
            $table->mediumText( 'text' );
            $table->unsignedBigInteger( 'user_id' );
            $table->unsignedInteger( 'post_id' );
            $table->unsignedMediumInteger( 'medium_int' );
            $table->unsignedSmallInteger( 'small_int' );
            $table->unsignedTinyInteger( 'tiny_int' );
            $table->text( 'body' );
            $table->tinyText( 'title' );

            $table->string( 'pinCode', 50 )->default( '1234' )->invisible();
            $table->timestamp( 'created_at' )->useCurrent();
            $table->timestamp( 'updated_at' )->useCurrent()-
>useCurrentOnUpdate();
```

migration command:

```
php artisan migrate
```

Renaming Table Name:

```
php artisan make:migration rename_profiles
```

```
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Migrations\Migration;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::rename( 'profiles', 'users' );
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        //
    }
};
```

migration command:

```
php artisan migrate
```

Dropping Table:

```
php artisan make:migration drop_greetings
```

```
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Migrations\Migration;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
```

```
    {
        Schema::dropIfExists( 'greetings' );
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        //
    }
};
```

migration command:

```
php artisan migrate
```

Adding a new column:

```
php artisan make:migration add_column_to_profiles
```

```
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::table( 'profiles', function ( Blueprint $table )
        {
            $table->after( 'name', function ( Blueprint $table )
            {
                $table->string( 'address' )->nullable();
                $table->string( 'phone' )->nullable()->unique();
            } );
        } );
    }

    /**
     * Reverse the migrations.
     */
```

```
    public function down(): void
    {
        Schema::table( 'profiles', function ( Blueprint $table )
        {
            //
        } );
    }
};
```

```
php artisan migrate
```

Renaming Column:

install this package for any column renaming version dependencies:

```
composer require doctrine/dbal
```

```
php artisan make:migration rename_column_to_profiles
```

```
public function up(): void
    {
        Schema::table( 'profiles', function ( Blueprint $table )
        {
            $table->renameColumn( 'name', 'fullName' );
        } );
    }
```

```
php artisan migrate
```

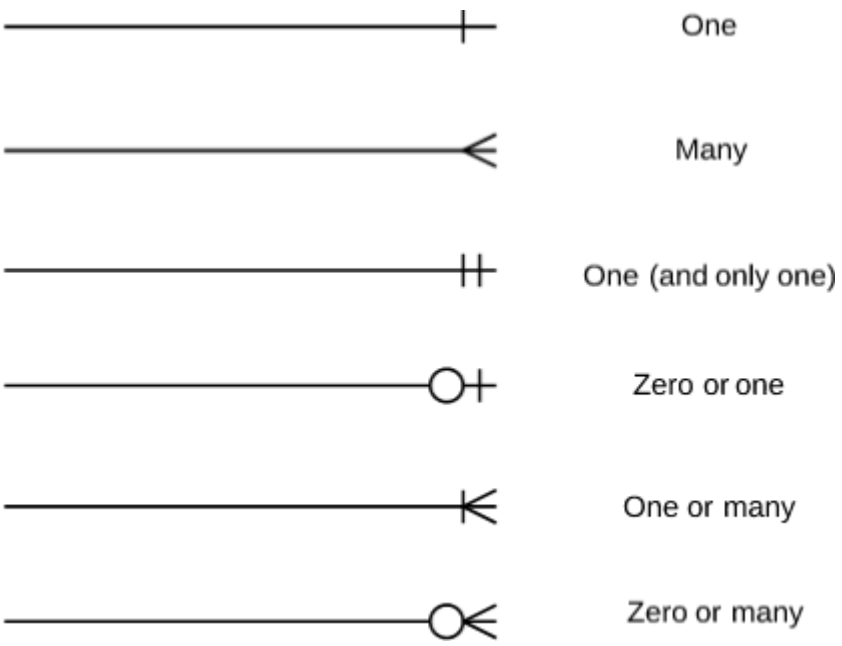
Drop a column:

```
php artisan make:migration drop_column_from_profiles
```

```
    public function up(): void
    {
        Schema::table( 'profiles', function ( Blueprint $table )
        {
            $table->dropColumn( 'fullName' );
            // can even drop multiple, for this, use an array
            // $table->dropColumn( ['address', 'phone'] );
        } );
    }
```

```
php artisan migrate
```

Relationships:



categories table:

```
Schema::create( 'categories', function ( Blueprint $table )
{
    $table->id();
    $table->string( 'categoryName', 50 );
    $table->string( 'categoryImage', 300 );
    $table->timestamps();
} );
```

brands table:

```
Schema::create( 'brands', function ( Blueprint $table )
{
    $table->id();
    $table->string( 'brandName', 50 );
    $table->string( 'brandImage', 300 );
    $table->timestamps();
} );
```

products table (relationship established with categories and brands table):

```
Schema::create( 'products', function ( Blueprint $table )
{
    $table->id();
    $table->string( 'title', 200 );
```

```
            $table->string( 'shortDescription', 500 );
            $table->string( 'price' );
            $table->boolean( 'discount' );
            $table->string( 'discountPrice', 50 );
            $table->string( 'image', 200 );
            $table->boolean( 'stock' );
            $table->float( 'star' );
            $table->enum( 'remark', ['new', 'hot', 'sale', 'popular', 'top'] );

            // foreign key declare
            $table->unsignedBigInteger( 'category_id' );
            $table->unsignedBigInteger( 'brand_id' );

            // relationship establish
            $table->foreign( 'category_id' )
                ->references( 'id' )
                ->on( 'categories' )
                ->restrictOnDelete()
                ->cascadeOnUpdate();

            $table->foreign( 'brand_id' )
                ->references( 'id' )
                ->on( 'brands' )
                ->restrictOnDelete()
                ->cascadeOnUpdate();

            $table->timestamps();
        } );
```

Query Builder:

To fetch all the data from a specific table of a database (get() method):

```
    public function index()
    {
        $result = DB::table( 'products' )->get();

        return $result;
    }
```

```
Route::get( '/', [DemoController::class, 'index'] );
```

To fetch a single row from a specific table of a database (first() method):

```
public function gettingFirstRowOnly()
{
    // will get only the data of first row of that table
    $result = DB::table( 'products' )->first();

    return $result;
}
```

To fetch a data of a specific id from a specific table of a database (find('id no') method):

```
public function gettingDataBasedOnId()
{
    // will get only the data of the given id of that table
    $result = DB::table( 'products' )->find( 18 );

    return $result;
}
```

To fetch all data of a specific column from a specific table of a database (pluck('column name') method):

```
public function gettingDataOfAnySpecificColumn()
{
    // will get only the data of the given column of that table
    $result = DB::table( 'products' )->pluck( 'name' );

    return $result;
}
```

```
public function gettingDataOfAnySpecificColumn()
{
    // we can also set another column as key after that column name
    $result = DB::table( 'products' )->pluck( 'name', 'id' );

    return $result;
}
```

To fetch multiple specific column's data from a specific table of a database (select('column names')->get() method):

```
public function selectClause()
{
    // this will return the data of the given columns
```

```
        $result = DB::table( 'products' )->select( 'id', 'name', 'price' )->get();

        return $result;
    }
```

To fetch multiple specific column's unique data from a specific table of a database (select('column name')->distinct()->get() method):

```
    public function distinctSelectClause()
    {
        // this will return the unique values of the given column
        $result = DB::table( 'products' )->select( 'unit' )->distinct()->get();

        return $result;
    }
```

Aggregates:

```
    public function aggregates()
    {
        // count() -> will return the total number of rows of this table
        $count = DB::table( 'products' )->count();

        // max() -> will return the maximum value of the given column
        $max = DB::table( 'products' )->max( 'price' );

        // min() -> will return the minimum value of the given column
        $min = DB::table( 'products' )->min( 'price' );

        // avg() -> will return the average value of the given column
        $avg = DB::table( 'products' )->avg( 'price' );

        // sum() -> will return the sum of the given column
        $sum = DB::table( 'products' )->sum( 'price' );

        return response( [
            'count' => $count,
            'max'   => $max,
            'min'   => $min,
            'avg'   => $avg,
            'sum'   => $sum,
        ] );
    }
```

Where Clause:

Where:

```
    public function whereClause()
    {
        // we can use ( =, !=, >, <, >=, <=, like, not like, in, not in ) type
operators
        $result = DB::table( 'products' )
            ->where( 'products.price', '=', '800' )
            ->get();

        return $result;
    }
```

```
    public function whereClause()
    {
        // we can use ( =, !=, >, <, >=, <=, like, not like, in, not in ) type
operators
        $result = DB::table( 'products' )
            ->where( 'products.price', '!=', '800' )
            ->get();

        return $result;
    }
```

```
    public function whereClause()
    {
        // we can use ( =, !=, >, <, >=, <=, like, not like, in, not in ) type
operators
        $result = DB::table( 'products' )
            ->where( 'products.price', '<', '800' )
            ->get();

        return $result;
    }
```

```
    public function whereClause()
    {
        // we can use ( =, !=, >, <, >=, <=, like, not like, in, not in ) type
operators
        $result = DB::table( 'products' )
            ->where( 'products.price', '>', '800' )
            ->get();
```

```
        return $result;
    }
```

```
    public function whereClause()
    {
        // we can use ( =, !=, >, <, >=, <=, like, not like, in, not in ) type
operators
        $result = DB::table( 'products' )
            ->where( 'products.price', '<=', '800' )
            ->get();

        return $result;
    }
```

```
    public function whereClause()
    {
        // we can use ( =, !=, >, <, >=, <=, like, not like, in, not in ) type
operators
        $result = DB::table( 'products' )
            ->where( 'products.price', '>=', '800' )
            ->get();

        return $result;
    }
```

```
    public function whereClause()
    {
        // we can use ( =, !=, >, <, >=, <=, like, not like, in, not in ) type
operators
        $result = DB::table( 'products' )
            ->where( 'products.price', '=', '800' )
            ->get();

        return $result;
    }
```

Where with LIKE operator:

```
    public function whereClause()
    {
        // LIKE operator will return the data that matches the given pattern
        $result = DB::table( 'products' )
            ->where( 'products.name', 'LIKE', '%ff%' )
            ->get();
```

```
        return $result;
    }
```

```
    public function whereClause()
    {
        // LIKE operator will return the data that matches the given pattern
        $result = DB::table( 'products' )
            ->where( 'products.name', 'LIKE', '%ff%' )
            ->get();

        return $result;
    }
```

Where With Not Like operator:

```
    public function whereClause()
    {
        // NOT LIKE will return the data which is not matching the given pattern
        $result = DB::table( 'products' )
            ->where( 'products.name', 'NOT LIKE', '%ff%' )
            ->get();

        return $result;
    }
```

Advanced Where Clause:

whereIn:

```
    public function whereClause()
    {
        // whereIn will return the data of the given column which is in the given
array
        $result = DB::table( 'products' )
            ->whereIn( 'products.price', [100, 200, 300] )
            ->get();

        return $result;
    }
```

whereNotIn:

```
    public function whereClause()
    {
        // whereNotIn will return the data of the given column which is not in
the given array
```

```
        $result = DB::table( 'products' )
            ->whereNotIn( 'products.price', [100, 200, 300] )
            ->get();

        return $result;
    }
```

orWhere:

```
    public function whereClause()
    {
        $result = DB::table( 'products' )
            ->where( 'products.price', '=', '200' )
            ->orWhere( 'products.price', '>', '300' )
            ->get();

        return $result;
    }
```

whereNot:

```
    public function whereClause()
    {
        $result = DB::table( 'products' )
            ->where( 'products.price', '=', '200' )
            ->whereNot( 'products.price', '>', '300' )
            ->get();

        return $result;
    }
```

whereBetween:

```
    public function whereClause()
    {
        $result = DB::table( 'products' )
            ->where( 'products.price', '=', '200' )
            ->whereBetween( 'products.price', [100, 300] ) // get the data
between 100 to 300
            ->get();

        return $result;
    }
```

whereNotBetween:

```
    public function whereClause()
    {
        $result = DB::table( 'products' )
            ->where( 'products.price', '=', '200' )
            ->whereNotBetween( 'products.price', [300, 800] ) // opposite of
whereBetween
            ->get();

        return $result;
    }
```

whereNull:

```
    public function whereClause()
    {
        $result = DB::table( 'products' )
            ->whereNull( 'products.price' )
            ->get();

        return $result;
    }
```

whereNotNull:

```
    public function whereClause()
    {
        $result = DB::table( 'products' )
            ->whereNotNull( 'products.price' )
            ->get();

        return $result;
    }
```

whereColumn:

```
    public function whereClause()
    {
        $result = DB::table( 'products' )
            ->whereColumn( 'products.updated_at', '=', 'products.created_at' )
            ->get();

        return $result;
    }
```

whereDay:

```
public function whereClause()
{
    $result = DB::table( 'products' )
        ->whereDay( 'products.updated_at', '=', '20' )
        ->get();

    return $result;
}
```

whereMonth:

```
public function whereClause()
{
    $result = DB::table( 'products' )
        ->whereMonth( 'products.updated_at', '=', '10' )
        ->get();

    return $result;
}
```

whereYear:

```
public function whereClause()
{
    $result = DB::table( 'products' )
        ->whereYear( 'products.updated_at', '=', '2023' )
        ->get();

    return $result;
}
```

whereTime:

```
public function whereClause()
{
    $result = DB::table( 'products' )
        ->whereTime( 'products.updated_at', '=', '22:53:03' )
        ->get();

    return $result;
}
```

whereDate:

```
public function whereClause()
{
```

```
        $result = DB::table( 'categories' )
            ->whereDate( 'categories.created_at', '=', '2023-07-23' )
            ->get();

        return $result;
    }
```

Join:

Inner Join:

```
    public function innerJoin()
    {
        $result = DB::table( 'products' )
            ->join( 'categories', 'products.category_id', '=', 'categories.id' )
            ->join( 'users', 'products.user_id', '=', 'users.id' )
            ->get();

        return $result;
    }
```

Left Join:

```
    public function leftJoin()
    {
        $result = DB::table( 'products' )
            ->leftJoin( 'categories', 'products.category_id', '=',
'categories.id' )
            ->leftJoin( 'users', 'products.user_id', '=', 'users.id' )
            ->get();

        return $result;
    }
```

Right Join:

```
    public function rightJoin()
    {
        $result = DB::table( 'products' )
            ->rightJoin( 'categories', 'products.category_id', '=',
'categories.id' )
            ->rightJoin( 'users', 'products.user_id', '=', 'users.id' )
            ->get();

        return $result;
    }
```

Cross Join:

```
    public function crossJoin()
    {
        // no need to provide the relationship column
        $result = DB::table( 'products' )
            ->crossJoin( 'categories' )
            ->crossJoin( 'users' )
            ->get();

        return $result;
    }
```

Advanced Join Clauses (with 'Where' condition):

```
    public function advancedJoin()
    {
        $result = DB::table( 'products' )
            ->join( 'categories', function ( JoinClause $join )
        {
                $join->on( 'products.category_id', '=', 'categories.id' )
                    ->where( 'products.price', '>', '2000' );
            } )
            ->join( 'users', function ( JoinClause $join )
        {
                $join->on( 'products.user_id', '=', 'users.id' )
                    ->where( 'products.price', '>', '2000' );
            } )
            ->get();

        return $result;
    }
```

Union:

```
    public function union()
    {
        $query1 = DB::table( 'products' )->where( 'products.price', '>', '200' );
        $query2 = DB::table( 'products' )->where( 'products.category_id', '=',
'34' );

        $result = $query1->union( $query2 )->get();

        return $result;
    }
```

Order Clause:

Ascending:

```
public function orderByClause()
{
    $result = DB::table( 'products' )
        ->orderBy( 'name', 'asc' )
        ->get();

    return $result;
}
```

Descending:

```
public function orderByClause()
{
    $result = DB::table( 'products' )
        ->orderBy( 'name', 'desc' )
        ->get();

    return $result;
}
```

inRandomOrder:

```
public function orderByClause()
{
    // randomized
    $result = DB::table( 'products' )
        ->inRandomOrder()
        ->get();

    return $result;
}
```

```
public function orderByClause()
{
    // randomized -> any one column
    $result = DB::table( 'products' )
        ->inRandomOrder()
        ->first();

    return $result;
}
```

Latest and Oldest:

Latest:

```
public function latest()
{
    $result = DB::table( 'products' )
        ->latest()
        ->get();

    return $result;
}
```

```
public function latest()
{
    $result = DB::table( 'products' )
        ->latest()
        ->first();

    return $result;
}
```

Oldest:

```
public function oldest()
{
    $result = DB::table( 'products' )
        ->oldest()
        ->get();

    return $result;
}
```

```
public function oldest()
{
    $result = DB::table( 'products' )
        ->oldest()
        ->first();

    return $result;
}
```

Skip, take():

```
public function skipAndTake()
{
    $result = DB::table( 'products' )
        ->skip( 5 )
        ->take( 5 )
        ->get();

    return $result;
}
```

Group By and Having (Having method works like where method):

Keep that in mind, group by and having method can show some trouble while getting data, to get rid off this problem , go to config -> database.php -> mysql -> 'strict' => true to false.

Group By:

```
public function groupBy()
{
    $result = DB::table( 'products' )
        ->groupBy( 'price' )
        ->get();

    return $result;
}
```

Having with groupBy:

```
public function having()
{
    $result = DB::table( 'products' )
        ->groupBy( 'price' )
        ->having( 'price', '>', '500' )
        ->get();

    return $result;
}
```

Inserting data in database:

```
public function insertMethod()
{
    // hard coded values
    $result = DB::table( 'users' )
        ->insert( [
            'firstName' => 'Hasan',
```

```
                'lastName'  => 'Mahmud',
            ] );

        return $result;
    }
```

```
    public function insertMethodFromRequestBodyAllAtOnce( Request $request )
    {
        // all at once
        // keep that in mind, for at once picking ($request->input()), input keys must be same as table column names in database
        $data = $request->input(); // (only json body)

        $result = DB::table( 'users' )
            ->insert( $data );

        return $result;
    }
```

```
    public function insertMethodFromRequestBodyOneByOne( Request $request )
    {
        // values picked from request body one by one
        $result = DB::table( 'users' )
            ->insert( [
                'firstName' => $request->input( 'firstName' ),
                'lastName'  => $request->input( 'lastName' ),
                'email'     => $request->input( 'email' ),
                'mobile'    => $request->input( 'mob' ),
                'password'  => $request->input( 'pass' ),
                'otp'       => $request->input( 'otp' ),
            ] );

        return $result;
    }
```

Updating data in database:

```
    public function updateMethodHardCoded()
    {
        // hard coded values
        $result = DB::table( 'users' )
            ->where( 'id', '=', '16' )
            ->update( [
                'firstName' => 'Hasan',
                'lastName'  => 'Mahmud',
            ] );
```

```
        return $result;
    }
```

```
    public function updateMethodPickIdFromUrlParameter( Request $request )
    {
        // Hard coded but picking id from url parameter
        $id = $request->id;

        $result = DB::table( 'users' )
            ->where( 'id', '=', $id )
            ->update( [
                'firstName' => 'Abul',
                'lastName'  => 'Hasan',
            ] );

        return $result;
    }
```

UpdateOrInsert():

```
    public function updateOrInsertMethod( Request $request )
    {
        // Define the attributes to check
        $attributes = [
            'id' => $request->id,
        ];

        // Define the values to update or insert
        $values = $request->input();

        // Perform update or insert
        $result = DB::table( 'products' )
            ->updateOrInsert( $attributes, $values );

        return $result;
    }
```

Increment:

```
    public function incrementMethod( Request $request )
    {
        $result = DB::table( 'products' )
            ->where( 'id', '=', $request->id )
            ->increment( 'price', 200 );
```

```
        return $result;
    }
```

Decrement:

```
    public function decrementMethod( Request $request )
    {
        $result = DB::table( 'products' )
            ->where( 'id', '=', $request->id )
            ->decrement( 'price', 1 );

        return $result;
    }
```

Delete Action:

```
    public function deleteMethod( Request $request )
    {
        $result = DB::table( 'users' )
            ->where( 'id', '=', $request->id )
            ->delete();

        return $result;
    }
```

```
Route::delete( '/30/{id}', [DemoController::class, 'deleteMethod'] );
```

Pagination:

```
    public function paginationMethod()
    {
        $result = DB::table( 'categories' )
            ->simplePaginate( 5 );

        return $result;
    }
```

Custom Pagination:

```
    public function customPaginationMethod()
    {
        $perPage  = 5;
        $pageName = 'item';
        $column   = ['*']; // * means all columns, we can define particular
column also
```

```
        $result = DB::table( 'categories' )
            ->paginate( $perPage, $column, $pageName );

        return $result;
    }
```

Laravel ORM (Object Relational Mapper):

For each table in database, there will be a particular model. As we know that, table name always will be plural, for ex. users, for this 'users' table, the model file name will be in singular - 'User'. For following this perfect naming convention, Laravel will automatically recognize the table of that model.

Naming Convention:

For table 'invoice_products', model will be 'InvoiceProduct':

```
php artisan make:model InvoiceProduct
```

For 'customers' table, model will be 'Customer':

```
php artisan make:model Customer
```

Hide column in model file, so that, while fetching data, that column will be hidden (protected $hidden):

Let's say in User model file:

```
    protected $hidden = [
        'password',
        'remember_token',
        'otp',
        'created_at',
        'updated_at',
    ];
```

show column in model file, so that, while fetching data, that column will be visible, rest will be hidden (protected $visible):

```
    // will show only selected data
    protected $visible = [
        'email',
    ];
```

Data getting/fetching from database table:

Retrieving all rows -> get():

```php
<?php

namespace App\Http\Controllers;

use App\Models\User;

class DemoController extends Controller
{
    public function dataGet()
    {
        /* previously we get the data by query builder like this:
        return DB::table( 'users' )->get();
        now for ORM, we will replace 'DB::table( 'users' )' with: User::get();
        DB::table( 'users' ) -> this extra line is not necessary now
         */

        return User::get();
    }
}
```

Retrieving all rows -> all():

```php
    public function dataGetAll()
    {
        return User::all();
    }
```

Finding a row by id -> find():

```php
    public function findMethod()
    {
        return User::find(2);
    }
```

Retrieving single row -> first() -> only single first row of table:

```php
    public function onlyOneRow()
    {
        return User::first();
    }
```

Retrieving list of columns -> pluck() -> only selected column list, not full data:

```
public function retrievingListOfColumns()
{
    return User::pluck( 'firstName' );
}
```

Another one:

```
public function retrievingListOfColumnsWithKeyDefining()
{
    return User::pluck( 'firstName', 'id' );
}
```

Data getting/fetching from database tables with relationships (with()):

Let's say user and category relationship, every user has a category, and category belongs to user. That means inside categories table there is a relation established with user by column user_id.

Now, types:

- **hasOne (one-to-one, one-and-only, one-to-zero relationships) -> only one relationship established data will be fetched of the other table**
- **hasMany (one-to-many, one-to-zero relationships) -> all will be fetched**
- **belongsTo (many-to-many relationships) -> when a data belongs to other table**

So, User has Category (many categories)
Category belongsTo User

so in User model:

```
public function category()
{
    return $this->hasMany( Category::class );
}

// another one
public function customer()
{
    return $this->hasMany( Customer::class );
}

// another one
public function product()
{
```

```
        return $this->hasMany( Product::class );
    }
```

Now get the users with categories (with() method):

```
    public function dataGet()
    {
        return User::with( 'category', 'customer', 'product' )->get();
    }
```

In Category model:

```
    public function user()
    {
        return $this->belongsTo( User::class );
    }
```

Now get the categories with users (with() method):

```
    public function dataGetCategories()
    {
        return Category::with( 'user' )->get();
    }
```

Data inserting in database:

For this, we must need to define the table column names, which columns will be fillable, let's say for 'users' table, we will work on 'User' model file and define this:

```
    protected $fillable = [
        'firstName',
        'lastName',
        'email',
        'mobile',
        'password',
        'otp'
    ];
```

We can also set, default values for any column, let's say 'otp', we can set default value here. If the 'otp' is given from user, then it will be saved like this, but if not then the default value will be set.

```
    protected $attributes = [
        'otp' => '0',
```

```
    ];
```

Let's insert:

```
    public function insertDataAllAtOnce( Request $request )
    {
        return User::create( $request->input() );
    }
```

To insert multiple at once:

But, keep that in mind, inserting multiple data all at once may sometimes create problem like, some data entered, some didn't. It will create database ACID violation. So, that we need to use try-catch block, DB::beginTransaction(), DB::commit() and DB::rollBack() methods.

```
    public function insertMultipleDataAllAtOnce( Request $request )
    {
        DB::beginTransaction(); // database session creation

        try {
            $data = $request->input();

            foreach ( $data as $value )
            {
                User::create( [
                    'firstName' => $value['firstName'],
                    'lastName'  => $value['lastName'],
                    'email'     => $value['email'],
                    'mobile'    => $value['mobile'],
                    'password'  => $value['password'],
                ] );
            }

            DB::commit(); // session will commit/confirm all actions

            return "ok";
        }
        catch ( Exception $exception )
        {
            DB::rollBack(); // session will rollback all actions if any error
occurs

            return "not ok";
        }
    }
```

Data updating in database:

```php
public function dataUpdate( Request $request )
{
    $id = $request->id;

    $data = $request->input();

    return User::where( 'id', '=', $id )->update( $data );
}
```

updateOrCreate():

```php
public function updateOrCreateMethod( Request $request )
{
    $attributes = [
        'firstName' => $request->firstName,
    ];

    $data = User::updateOrCreate(
        $attributes,
        $request->input()
    );

    return $data;
}
```

Data deleting in database:

```php
public function dataDelete( Request $request )
{
    $id = $request->id;

    return User::where( 'id', '=', $id )
        ->delete();
}
```

Increment:

```php
public function incrementMethod( Request $request )
{
    $id = $request->id;

    $result = Product::where( 'id', '=', $id )
        ->increment( 'price', 500 );

    return $result;
}
```

Decrement:

```
    public function decrementMethod( Request $request )
    {
        $id = $request->id;

        $result = Product::where( 'id', '=', $id )
            ->decrement( 'price', 300 );

        return $result;
    }
```