

Laravel

1. Global installation

- composer global require laravel/installer
- laravel new example-app

2. per project installation

- composer create-project laravel/laravel example-app

3. project run

- php artisan serve

4. project run stop

-ctr+c

5. Laravel Folder & File Structure

- Model Folder -> Database/sql queries Handling Files (app->models)
- controller Folder -> Business Logics Files (app->http->controller)
- view Folder ->Html files (resources->views)
- Routing Folder -> URI defining Files
- Asserts Folder(public Folder) -> images/fonts/music/videos/css/javascript Files

6. Route---->URI

- webpage URL

7.Search any file vscode

-ctr+p

8.All command show

-php artisan

9.controller create

-php artisan make:controller (Name)

10.table create

-php artisan make:migration create_tableName_table

11.Table name create of sql

-php artisan migrate

12.last update table k delete kore dibe

-php artisan migrate:rollback

13.database empty

-php artisan migrate:reset

14.database refresh

-php artisan migrate:refresh

15.model and table create same time

-php artisan make:model (Name) -m

1.Routing

```
1. Route::get('/', function () {  
    return view('welcome');  
});  
  
2. Route::get('/post', function () {  
    return view('post');  
});  
  
3.single line
```

```
Route::view('/post','post');
```

Laravel Routing Parameters & Constraints

```
Route::get('/post/{id}', function (string $value) {  
    return "<h1>Post Id :". $value . "</h1>";  
});
```

optional parameter

```
Route::get('/post/{id?}', function (string $value = null) {  
    if($value){  
        return "<h1>Post Id :". $value . "</h1>";  
    }else{  
        return "<h1> No id found </h1>";  
    }  
});
```

```
});
```

multi parameter passing

```
Route::get('/post/{id?}/comment/{commentid?}', function (string $value = null, string $comment = null) {
    if($value){
        return "<h1>Post Id :". $value ."</h1> <h2>". $comment ."</h2>";
    }else{
        return "<h1> No id found </h1>";
    }
});
```

laravel Constraints

```
Route::get('/post/{id?}', function (string $value = null) {
    if($value){
        return "<h1>Post Id :". $value ."</h1>";
    }else{
        return "<h1> No id found </h1>";
    }
});->whereNumber\('id'\); //id== number hobe
```

```
Route::get('/post/{id?}', function (string $value = null) {
    if($value){
        return "<h1>Post Id :". $value ."</h1>";
    }else{
        return "<h1> No id found </h1>";
    }
})->whereIn('id',['moive','song','painting']); // id value ei tin tar modder value dile
kaj korbe noy parameter kaj korbe na
```

```
Route::get('/post/{id?}', function (string $value = null) {
    if($value){
        return "<h1>Post Id :". $value ."</h1>";
    }else{
        return "<h1> No id found </h1>";
    }
})->where('id','[0-9]+'); // id value regular expersion er modder dite hobe
```

```
Route::get('/post/{id?}', function (string $value = null) {
    if($value){
        return "<h1>Post Id :". $value ."</h1>";
    }else{
        return "<h1> No id found </h1>";
    }
})
```

```
}->where('id','[a-zA-Z]+'); // id value Aphavetic regular expersion er modder dite  
hobe
```

```
Route::get('/post/{id?}/comment/{commentid?}', function (string $value =  
null,string $comment =null) {  
    if($value){  
        return "<h1>Post Id :". $value ."</h1> <h2>". $comment ."</h2>";  
    }else{  
        return "<h1> No id found </h1>";  
    }  
}
```

```
}->where('id','[a-zA-Z]+')->whereAlpha('commentid');
```

Laravel Named Route

```
Route::get('/post/about-us', function () {  
    return view('post');  
})->name('about');
```

```
// <a href='{{route('about)}}'>Post</a> jokhon post click korbo tokhon about-us  
route korbe
```

Redirect route

```
Route::redirect('/about','/test'); // user about route click korle autometric test  
route niye jabe
```

```
Route::redirect('/about','/test',301);
```

```
// 301->permanent redirect code
```

Routes Group

```
// url-> /page/....
```

```
Route::prefix('page')->group(function (){
```

```
    Route::get('/about-us', function () {
```

```
        return view('post');
```

```
    });
```

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

```
        return view('post');
```

```
    });
```

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

```
        return view('post');
```

```
    });
```

```
});
```

Not found page Route

```
Route::fallback(function(){
    return "<h1>Page Not found</h1>";
})
```

Blade Template Main Directives

1. @include -> jeta korbo sei file add hobe
2. @section
3. @extend
4. @yield

include

```
@include('pages.header')

@include('pages.header',['name'=>'aminur'])

@include('pages.footer')

jodi na thake page taw error dekhabe na include if

@ includelf('pages.footer')

@@includeWhen(condition, 'view.name', ['some' => 'data']) condition true hole
view page show korbe
```

Layouts Using Template Inheritance

Defining a Layout

```
<!-- resources/views/layouts/app.blade.php -->
```

```
<html>
```

```
<head>
    <title>App Name - @yield('title')</title>
</head>
<body>
    @section('sidebar')
        This is the master sidebar.
    @show

    <div class="container">
        @yield('content')
    </div>
</body>
</html>
```

Extent layout

```
<!-- resources/views/child.blade.php -->
```

```
@extends('layouts.app')

@section('title', 'Page Title')

@section('sidebar')
    @parent
```

```
<p>This is appended to the master sidebar.</p>
@endsection
```

```
@section('content')
<p>This is my body content.</p>
@endsection
```

Php convert json

```
@php
$user = "Aminur";
$fruits = ['Apple', "Orange", "Banana", "Grapes"];
@endphp
```

```
<script>
var data = @json($fruits);
data.forEach(element => {
    console.log(element)
});
</script>
```

Defining a Layout

```
<!-- resources/views/layouts/app.blade.php -->
```

```
<html>
```

```
<head>
    <title>App Name - @yield('title')</title>
</head>
<body>
    @section('sidebar')
        This is the master sidebar.
    @show

    <div class="container">
        @yield('content')
    </div>
    @stack('javascript') //javascript addd
</body>
</html>
```

Extent layout

```
<!-- resources/views/child.blade.php -->
```

```
@extends('layouts.app')
```

```
@section('title', 'Page Title')
```

```
@section('sidebar')
```

```
@parent

<p>This is appended to the master sidebar.</p>

@endsection

@section('content')
    <p>This is my body content.</p>
@endsection

@push('javascript')
    Jslink add korbo
@endpush
```

Laravel Pass Data Route to View

Route define

```
Route::get('/view', function () {
    $name = 'Yahoo Baba';
    return view('post',['user'=>$name]);
});

Route::get('/view', function () {
    $name = 'Yahoo Baba';
    return view('post',['user'=>$name,'city'=>'delhi']);
});
```

View file execute

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

Controller Tutorial

1.controller file create `php artisan make:controller(Name)`

2.controller classs create

3.route create

Control page

```
class pageController extends Controller
```

```
{
```

```
    public function showUser(){
```

```
        return view('welcome');
```

```
}
```

```
}
```

Route page

```
use App\Http\Controllers\pageController;
```

```
Route::get('/',[pageController::class,'showUser']);
```

```
class pageController extends Controller
```

```
{
```

```
    public function showUser(String $id){
```

```
        return view('welcome',['id'=>$id]);
```

```
    }
}

Route::get('/user/{id}',[pageController::class,'showUser']);
```

Group controller in route

```
Route::Controller(pagecontroller::class)->group(function(){
    Route::get('/', 'showUser');
    Route::get('/user/{id}', 'showUser');
});
```

Database Migration

Step:1 create database

-.env file database setup korte hobe

Step:2 create database migration(create tables)

```
1.php artisan make:migration create_tableName_table
```

2.add to table row name

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

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

```
    $table->string('name');
```

```
    $table->string('email');
```

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

```
});
```

3.php artisan migrate

Step:3 seeding(insert initial data table)

Step 4:create model

*update column

- 1.command
- 2.add column
- 3.php artisan migrate

Step to work in Seeder

1. Php artisan make:model databasetable
2. Php artisan make:seeder ModelNameSeeder
- 3.use App/Models/student;

```
Class StudentSeeder extends Seeder{
```

```
Public function run():void{
```

```
Student::create([
```

```
‘name’=>’Yahoo Baba’,
```

```
‘email=>’noor@gmail.com’
```

```
]);
```

```
}
```

```
}
```

```
4.Seeders/DatabaseSeeder.php
```

```
$this->call([
```

```
StudentSeeder::class
```

```
]);
```

```
5.php artisan db:seed
```

Note:database name somoy s dite hobe r baki kkhete s dite hob na

Mutiple data add

```
public function run(): void
{
    $students =collect(
        [
            [
                'name'=>'a',
                'email'=>'a@gmail.com'
            ],
            [
                'name'=>'b',
                'email'=>'b@gmail.com'
            ],
            [
                'name'=>'c',
                'email'=>'c@gmail.com'
            ],
            [
                'name'=>'d',
                'email'=>'d@gmail.com'
            ],
        ]
    );
}
```

```
$students->each(function($student){  
    student::insert($student);  
});  
  
}  
}
```

Json file theke data add kora

Json file

```
[  
    {  
        "name": "aminur",  
        "email": "aminur@gmail.com"  
    },  
    {  
        "name": "a",  
        "email": "a@gmail.com"  
    },  
    {  
        "name": "b",  
        "email": "b@gmail.com"  
    },  
]
```

```
{  
    "name": "c",  
    "email": "c@gmail.com"  
,  
{  
    "name": "d",  
    "email": "d@gmail.com"  
}  
]  
]
```

Seeder file

```
public function run(): void  
{  
  
    $json = File::get(path:'database/json/students.json');  
    $students = collect(json_decode($json));  
    $students->each(function($student){  
        student::create(  
    [  
        ]  
    ]  
})  
}]
```

```
'name'=>$student->name,  
'email'=>$student->email  
  
]  
);  
});  
  
}
```

Again data add time this command

```
Php artisan migrate:fresh --seed
```

Fake data implementation

```
public function run(): void  
{  
    student::create([  
        'name'=>fake()->name(),  
        'email'=>fake()->unique()->email()  
    ]);  
}
```

Multiple fake data implementation

```
public function run(): void  
{
```

```
for($i=1;$i<=10;$i++){
    student::create([
        'name'=>fake()->name(),
        'email'=>fake()->unique()->email()
    ]);
}
}
```

Steps to work in Factory (fake data add ,can not using loop only using function then create a multiple fake data)

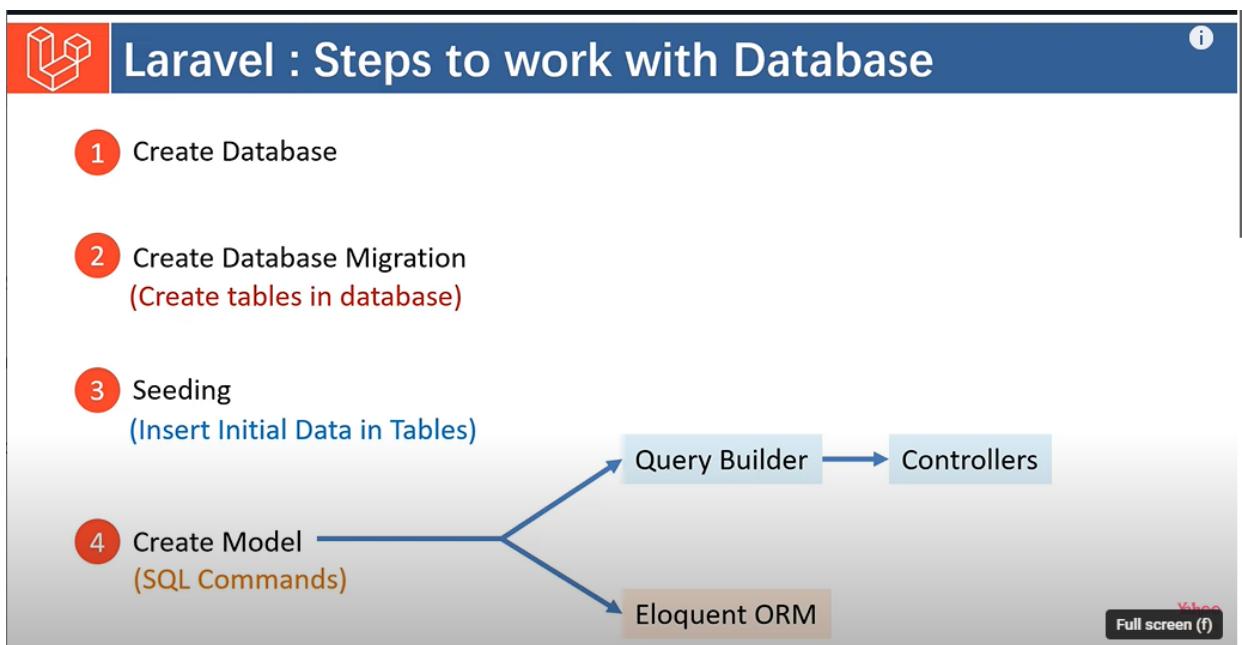
- 1.php artisan make:model ModelName
- 2.php artisan make:factory ModelNameFactory
- 3.class StudentFactory extends Factory{
 Public function definition():array{
 Return[
 'name'=>fake()->name(),
 'email'=>fake()->email()
];
 }
}
- 4.Seeders/DatabaseSeeder.php
 Use App\Models\student;
 Student::factory()->count(5)->create();

Note: count holo kot gulo data create korbe seta

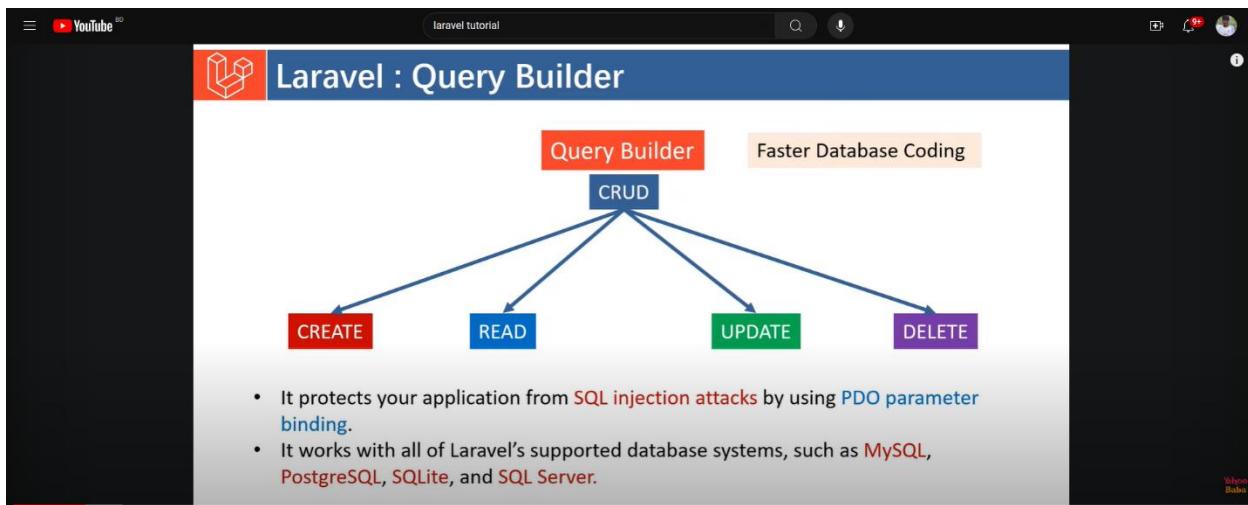
5.php artisan db:seed

Fake data documentation

<https://fakerphp.org/>



Laravel Query Builder



Laravel : Steps to work in Query Builder

- 1 `php artisan make:controller UserController`
- 2

```
use Illuminate\Support\Facades\DB;

class UserController extends Controller
{
    public function show()
    {
        $users = DB::table('users')->get();
        return $users;
    }
}
```
- 3

```
use App\Http\Controllers\UserController;

Route::get('/user', [UserController::class, 'show']);
```

- Read – `get()`
- Insert – `insert()`
- Update – `update()`
- Delete – `delete()`

<http://localhost/user>

Read data with query Builder



Laravel : Read Data with Query Builder

```
SELECT * FROM users
```

```
DB::table('users')->get()
```

```
SELECT name, city FROM users
```

```
DB::table('users') ->select('name', 'city')->get()
```

```
SELECT * FROM users WHERE city = 'goa';
```

```
DB::table('users') ->where('city', '=', 'goa')->get()
```

```
DB::table('users') ->where('city', '=', 'goa') ->where('age', '>', 18)->get()
```

```
DB::table('users') ->where('city', '=', 'goa') ->orWhere('age', '>', 18)->get()
```

- whereBetween()
- whereIn()
- whereNull()
- whereMonth()
- whereDay()
- whereYear()
- whereTime()

Crontroller create

```
-php artisan make:controller UserController
```

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;

class UserController extends Controller
{
    public function showUsers(){
        $users =DB::table('users')->get();
        return $users;
    }
}
```

Route

```
<?php

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

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

Route::get('/', [UserController::class, 'showUsers']);
```

2.controller

```
Http > Controllers > UserController.php > UserController > showUsers
<?php

namespace App\Http\Controllers;

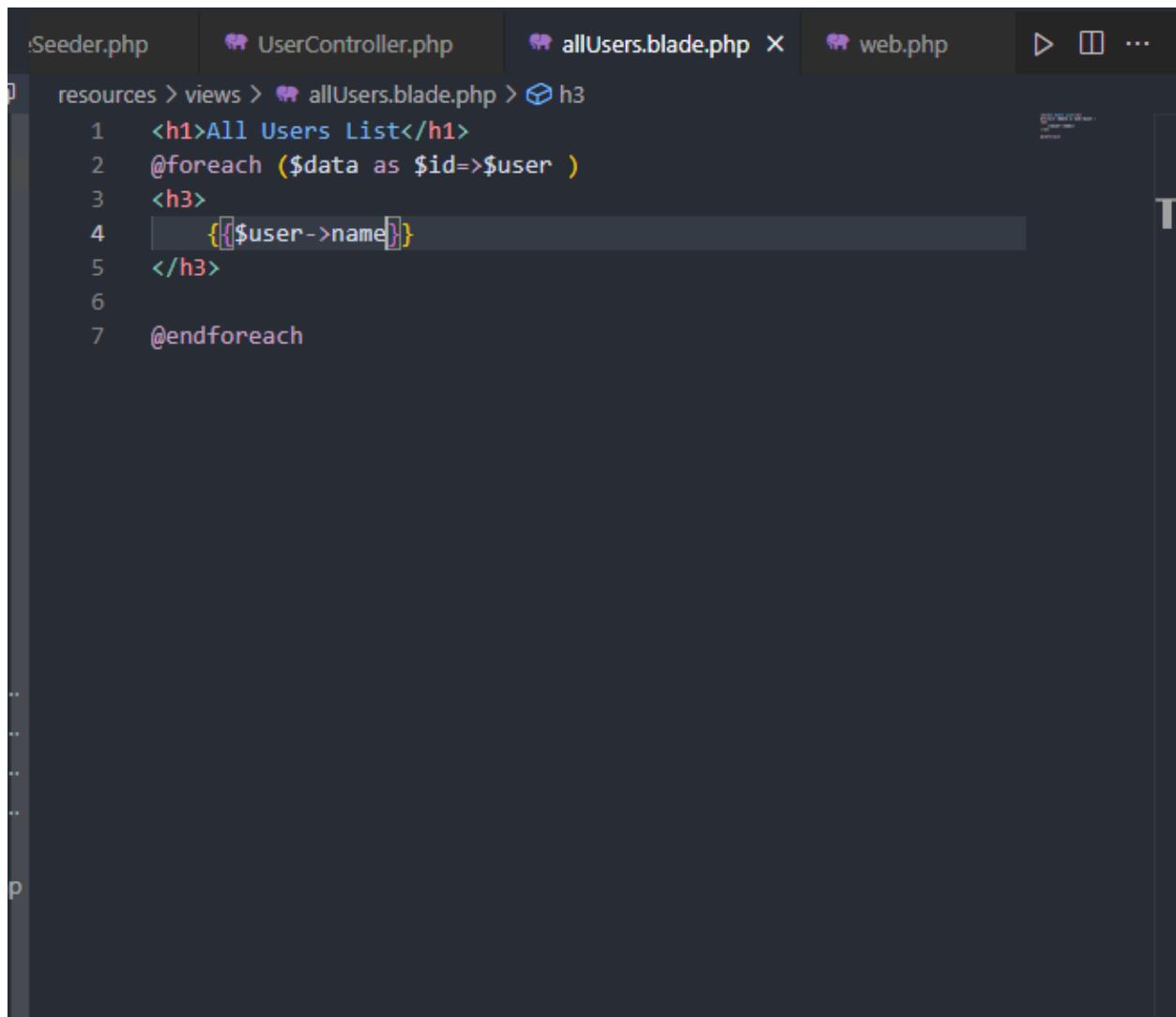
use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;

class UserController extends Controller
{
    public function showUsers(){
        $users =DB::table('users')->get();
        // return $users;
        return view ('allUsers',['data'=>$users]);
    }
}
```

Route

```
Route::get('/',[UserController::class,'showUsers']);
```

View



The screenshot shows a code editor interface with a dark theme. At the top, there are tabs for 'Seeder.php', 'UserController.php', 'allUsers.blade.php' (which is the active tab), and 'web.php'. Below the tabs, the file content is displayed:

```
resources > views > allUsers.blade.php > h3
1  <h1>All Users List</h1>
2  @foreach ($data as $id=>$user )
3  <h3>
4  |   {{$user->name}}
5  </h3>
6
7  @endforeach
```

3.

The screenshot shows a code editor with several tabs at the top: UserSeeder.php, DatabaseSeeder.php, UserController.php (active), web.php, user.json, and others. The UserController.php tab shows the following PHP code:

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use Illuminate\Support\Facades\DB;
7
8 class UserController extends Controller
9 {
10     public function showUsers(){
11         $users =DB::table('users')->get();
12         // return $users;
13         return view ('allUsers',[ 'data'=>$users]);
14     }
15     public function singleUser(string $id){
16         $users =DB::table('users')->where('id',$id)->get();
17         // return $users;
18         return view ('allUsers',[ 'data'=>$users]);
19     }
20 }
21
```

```
Route::get('/',[UserController::class,'showUsers']);
Route::get('/user/{id}',[UserController::class,'singleUser'])->name('view.user');
```

//button

```
<td><a href="{{ route('view.user', $user->id) }}>" class="btn
/tr>
```

//column select

```
public function showuser(){
    $user=DB::table('users')->select('name','email')->get();
    return $user;
}
```

Laravel : Query Builder

use Illuminate\Support\Facades\DB;

Add New Record

```
DB::table('users')->insert([
    'name' => 'Yahoo Baba',
    'email' => 'yahoobaba@email.com',
]);
```

Delete Record

```
DB::table('users')
->where('id', 1)
->delete();
```

Update Existing Record

```
DB::table('users')
->where('id', 1)
->update([
    'city' => 'Agra',
]);
```

```
public function addUser(){
    $user = DB::table('users')->insert([
        'name'=>'Aminur',
        'email'=>'noor@gmail.com',
        'password'=>'12345678'

    ]);
    if($user){
        echo "<h1>Data Successfully Added.</h1>";
    }
}
```

```
Route::get('/add',[UserController::class,'addUser']);
```

Multiple value insert

```
public function addUser(){
    $user = DB::table('users')->insert([
        [
            'name'=>'d',
            'email'=>'d@gmail.com',
            'password'=>'12345678'
        ],
        [
            'name'=>'a',
            'email'=>'a@gmail.com',
            'password'=>'12345678'
        ],
        [
            'name'=>'b',
            'email'=>'b@gmail.com',
            'password'=>'12345678'
        ]
    ]);
    if($user){
        echo "<h1>Data Successfully Added.</h1>";
    }
}
```

Check unique email

```
public function addUser(){
    $user = DB::table('users')->insertOrIgnore([
        [
            'name'=>'d',
            'email'=>'d@gmail.com',
            'password'=>'12345678'
        ]
    ]);
    if($user){
        echo "<h1>Data Successfully Added.</h1>";
    }else{
        echo "<h1>Data not Added.</h1>";
    }
}
```

Update

```
public function updateUser(){
    $user = DB::table('users')->where('id',1)->update(['name'=>'noor']);
}
```

```
Route::get('/update',[UserController::class,'updateUser']);
```

Delete

```
public function deleteUser(){
    $user = DB::table('users')->where('id',1)->delete();
}
```

Query Builder with Html Forms

The screenshot shows a Laravel application interface. On the left, there is a form with fields for 'Name' and 'Email', and a 'Submit' button. On the right, the code for the form is displayed:

```
<form action="/addUser" method="POST">
    @csrf
    <input type="text" name="username">
    <input type="text" name="userpassword">
    <input type="submit">
</form>
```

Below the form, there is a red box containing the text "Cross-site Request Forgery" and "malicious exploit". At the bottom, there is a navigation bar with links like "Home", "About", "Contact", and "Logout".

Controller

```
public function addUser(Request $req){
    $user = DB::table('users')->insert([
        'name'=>$req->username, //form input field er name je thakbe setai dite hobe
        'email'=>$req->useremail,
        'password'=>$req->userpassword

    ]);
    if($user){
        echo "<h1>Data Successfully Added.</h1>";
    }else{
        echo "<h1>Data not Added.</h1>";
    }
}
```

//route

```
Route::post('/add',[UserController::class,'addUser'])->name('addUser');
Route::view('newuser','/adduser');
```

//form

```
<body>
    <div class="container">
        <h1>Add New User</h1>
        <form action="{{route('addUser')}}" method="POST">
            @csrf
            <input type="text" placeholder="Enter the Name" name="username">
            <input type="email" placeholder="Enter the Email" name="useremail">
            <input type="password" placeholder="Enter the password" name="userpassword">
            <button type="submit">submit</button>
        </form>
    </div>
</body>
```

Update

```
public function updateUser(Request $req, $id){  
    $user = DB::table('users')  
        ->where('id', $id)  
        ->update([  
            'name' => $req->username,  
            'email' => $req->useremail,  
            'age' => $req->userage,  
            'city' => $req->usercity,  
        ]);  
  
    if($user){  
        echo "<h1>Data Successfully Updated.</h1>";  
    }else{  
        echo "<h1>Data not Updated.</h1>";  
    }  
}
```

1.first route->form theke data control kore

2.second route->home page update page er niye jay

```
Route::post('/update/{id}', 'updateUser')->name('update.user');  
Route::get('/updatepage/{id}', 'updatePage')->name('update.page');
```

```
<div class="container">  
    <div class="row">  
        <div class="col-4">  
            <h1>Update User Data</h1>  
            <form action="{{ route('update.user', $data->id) }}" method="POST">  
                @csrf  
                <div class="mb-3">  
                    <label class="form-label">Name</label>  
                    <input type="text" value="{{ $data->name }}" class="form-control" name="username">  
                </div>  
                <div class="mb-3">  
                    <label class="form-label">Email</label>  
                    <input type="text" value="{{ $data->email }}" class="form-control" name="useremail">  
                </div>  
                <div class="mb-3">  
                    <label class="form-label">Age</label>  
                    <input type="text" value="{{ $data->age }}" class="form-control" name="userage">  
                </div>
```

Pagination

The screenshot shows a section titled "Laravel : Pagination Methods". It lists three methods: 1. Paginate(), 2. simplePaginate(), and 3. cursorPaginate(). Each method has a corresponding code example and a Blade template snippet. The Blade template for simplePaginate() shows the {{ \$data->links() }} directive.

Blade File :

```
Blade File :  
{{ $data->links() }}
```

```
public function showUsers(){  
    $users = DB::table('users')  
        ->simplePaginate(4);  
  
    return view('allusers', ['data' => $users]);  
}
```

Show the pagination button

```
<div class="mt-5">  
    {{ $data->links() }}  
</div>
```

Urlbar

localhost:8000/?sort=votes&test=abc&page=2

```
public function showUsers(){
    $users = DB::table('users')
        ->paginate(5)
        ->appends(['sort' => 'votes', 'test' => 'abc']);
    //    return $users;
    return view('allusers', ['data' => $users]);
}
```

Note appends kaj korte hobe shudu

url bar hash value add



The screenshot shows a browser window with three tabs: 'All Users', 'Database: Pagination - Laravel', and 'Bootstrap - The most popular HT'. The address bar displays 'localhost:8000/?page=1#users'. The main content area is titled 'All Users List' and contains the following code:

```
public function showUsers(){
    $users = DB::table('users')
        ->paginate(5)
        ->fragment('users');
    //    return $users;
    return view('allusers', ['data' => $users]);
}
```

Note hash value jonno shudu fragment add korte hobe

Join Table



Laravel : Join Table

Students Table			
Id	Name	Age	City
1	Ram Kumar	19	1
2	Salman Khan	18	3
3	Meera Khan	19	2
4	Sarita Kumari	21	1

Cities Table

Cid	City
1	Agra
2	Bhopal
3	Delhi

```
SELECT *  
FROM students  
INNER JOIN cities  
ON students.city = cities.cid;
```

```
DB::table('students')  
->join('cities','students.city','=','cities.cid')  
->get();
```



Laravel : MySQL Joins

- INNER JOIN `join()`
- LEFT JOIN `leftJoin()`
- RIGHT JOIN `rightJoin()`
- CROSS JOIN `crossJoin()`

```
public function showStudents(){
    $students = DB::table('students')
        ->join('cities','students.city','=','cities.id')
        ->get();

    return $students;
}
```

```
Route::get('/',[StudentController::class,'showStudents']);
```

Select using

```
public function showStudents(){
    $students = DB::table('students')
        ->join('cities','students.city','=','cities.id')
        ->select('students.*','cities.city_name')
        ->get();

    // return $students;

    return view('welcome',compact('students'));
}
```

Left join

```

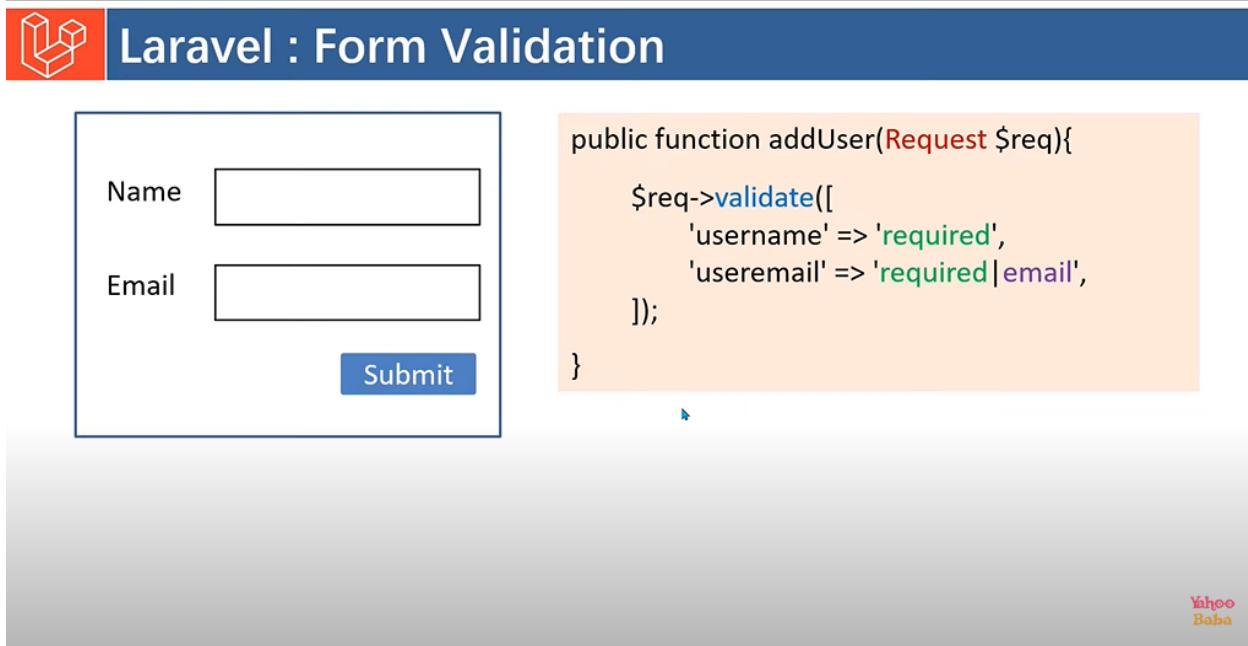
$students = DB::table('students')
    ->leftJoin('cities', function(JoinClause $join){
        $join->on('students.city', '=', 'cities.id')
            ->where('students.name', 'like', 'a%');
    })
    ->select('students.*', 'cities.city_name')
    ->get();

// return $students;

return view(['welcome', compact('students')]);

```

Form Validation



Laravel : Form Validation

Name

Email

Submit

```

public function addUser(Request $req){
    $req->validate([
        'username' => 'required',
        'useremail' => 'required|email',
    ]);
}

```

Error message

```
<h1>Add New User</h1>
@if ($errors->any())
    <ul class="alert alert-danger">
        @foreach ($errors->all() as $error)
            <li>{{ $error }}</li>
        @endforeach
    </ul>
@endif
```

Single error message show

```
<div class="mb-3">
    <label class="form-label">Name</label>
    <input type="text" class="form-control" name="username">
    <span class="text-danger">
        @error('username')
            {{ $message }}
        @enderror
    </span>
</div>
```

Invalid error show

Add New User

Name

Email



The useremail field is required.

Age

City

Submit

```
<div class="mb-3">
    <label class="form-label">Name</label>
    <input type="text" class="form-control @error('username') is-invalid @enderror" name="username">
    <span class="text-danger">
        @error('username')
            {{ $message }}
        @enderror
    </span>
</div>
```

<https://laravel.com/docs/11.x/validation#main-content>

validation attributes change

Email



The Email Address field must be a valid email address.

A screenshot of Visual Studio Code showing the file validation.php. The code defines a custom validation attribute for 'useremail' with the message 'Email Address'. The file path is vendor/laravel/framework/src/Illuminate/Translation/lang/en/validation.php.

```
172     /*
173      |--------------------------------------------------------------------------
174      | Custom Validation Attributes
175      |--------------------------------------------------------------------------
176      |
177      | The following language lines are used to swap our attribute placeholder
178      | with something more reader friendly such as "E-Mail Address" instead
179      | of "email". This simply helps us make our message more expressive.
180      |
181     */
182
183     'attributes' => [
184         'useremail' => 'Email Address',
185         'useremail' => 'Email Address',
186     ],
187
188 ];
189
```

Error required message show

A screenshot of Visual Studio Code showing the file UserController.php. It contains a validate method with rules for username, useremail, userpass, userage, and usercity fields. The validation messages are defined in the validation.php file shown above.

```
10     public function addUser(Request $req){
11
12         $req->validate([
13             'username' => 'required',
14             'useremail' => 'required|email',
15             'userpass' => 'required|alpha_num|min:6',
16             'userage' => 'required|numeric|min:18',
17             'usercity' => 'required',
18         ], [
19             "username.required" =>'User Name is required!',
20             "useremail.required" =>'User Email is required!',
21             "useremail.email" =>'Enter the correct email address.',
22             "userage.required" =>'User age is required.',
23             "userage.numeric" =>'User age must be numeric.',
24             "userage.min:18" =>'User age should not less than 18 years old.',
25             "usercity.required" =>'User is required.',
26         ]);
27
28     return $req->all();
29
```

Form Request Validation

The screenshot shows a YouTube video player with the title "Laravel : Form Request Validation". The video content displays three numbered steps of code:

1. `php artisan make:request UserRequest`
2.

```
class UserRequest extends FormRequest
{
    public function rules()
    {
        return [
            'username' => 'required',
            'useremail' => 'required|email'
        ];
    }
}
```
3.

```
public function addUser(UserRequest $req)
{
    return $req->all();
}
```

The code is presented in three colored boxes: light blue for the artisan command, light green for the UserRequest class definition, and light orange for the addUser method. The video player interface includes a search bar, a video progress bar showing 4:35 / 33:20, and a URL bar at the bottom.

Note: controller class er UserRequest import korbo

Message show

```
project-validation
File Edit Selection View Go Run Terminal Help
web.php adduser.blade.php UserController.php UserRequest.php validation.php
app > Http > Requests > UserRequest.php > UserRequest > messages
25     'username' => 'required',
26     'useremail' => 'required|email',
27     'userpass' => 'required|alpha_num|min:6',
28     'userage' => 'required|numeric|min:18',
29     'usercity' => 'required',
30 ];
31 }
32
33 public function messages(){
34     return [
35         "username.required" =>'User Name is required!',
36         "useremail.required" =>'User Email is required!',
37         "userpass.required" =>'User Password is required!',
38         "useremail.email" =>'Enter the correct email address.',
39         "userage.required" =>'User age is required.',
40         "userage.numeric" =>'User age must be numeric.',
41         "userage.min:18" =>'User age should not less than 18 years old.',
42         "usercity.required" =>'User city is required.',
43     ];
44 }
45
46 public function attributes(){
47     return [
48         'username' => 'User Name',
49         'useremail' => 'User Email',
50         'userpass' => 'User Password',
51         'userage' => 'User Age',
52         'usercity' => 'User City',
53     ];
54 }
55
56
```

13:56 / 33:20 • Make Message Method in Request Class >

Attributes

```
project-validation
File Edit Selection View Go Run Terminal Help
web.php adduser.blade.php UserController.php UserRequest.php validation.php
app > Http > Requests > UserRequest.php > UserRequest > attributes
41     "userage.min:18" =>'User age should not less than 18 years old.',
42     "usercity.required" =>'User city is required.',
43 ];
44 }
45
46 public function attributes(){
47     return [
48         'username' => 'User Name',
49         'useremail' => 'User Email',
50         'userpass' => 'User Password',
51         'userage' => 'User Age',
52         'usercity' => 'User City',
53     ];
54 }
55
56
```

18:21 / 33:20 • Make Attributes Method in Request Class >

Only one error message show

Add New User

Name

 I ?

The User Name field is required.

Email

Password

Age

City

 Delhi

Submit

```
protected $stopOnFirstFailure = true;  
}
```

Custom Validation Rule



Laravel : Custom Validation Rules

i

Custom Validation Rules

Using Rule Objects

Use Closure

Can Use many time in a website

Can Use one time in a website



Yahoo
Baba

Resource controller

☰ YouTube BD

Search



Laravel : Type of Controllers

i

1 Basic Controller

```
php artisan make:controller UserController
```

2 Single Action Controller

```
php artisan make:controller UserController --invokable
```

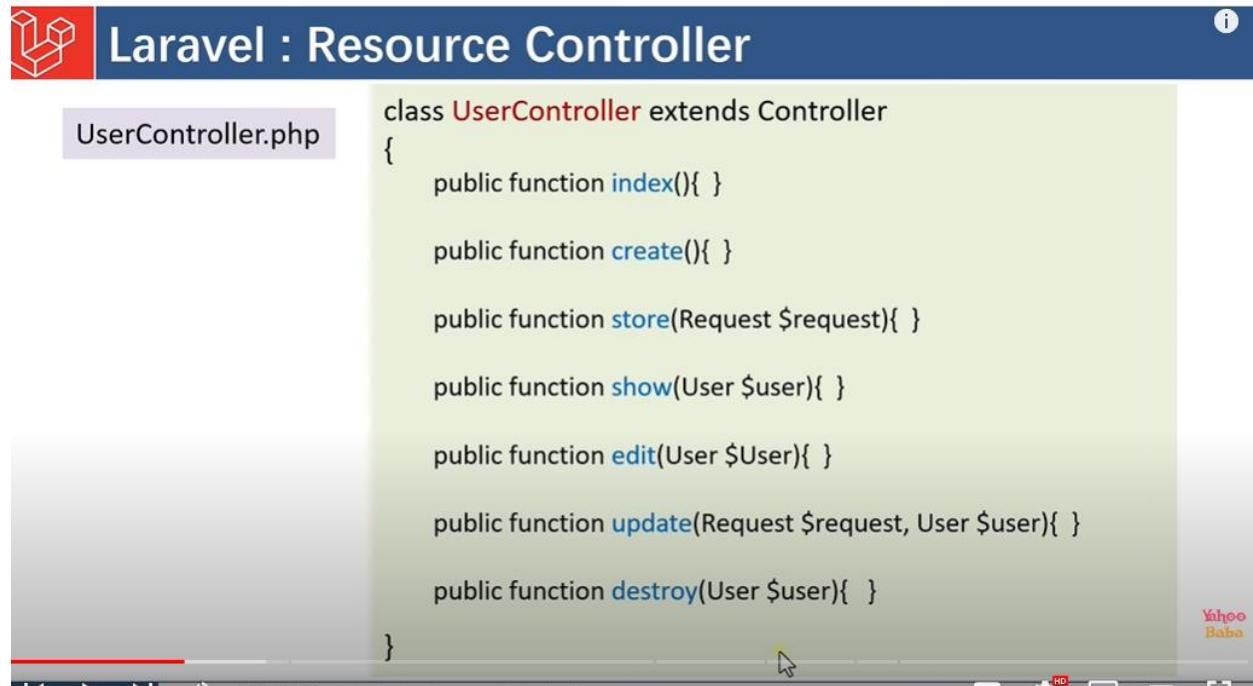
3 Resource Controller (Create, Read, Update, Delete)

```
php artisan make:controller UserController --resource
```

Yahoo
Baba

Note automatic create many function

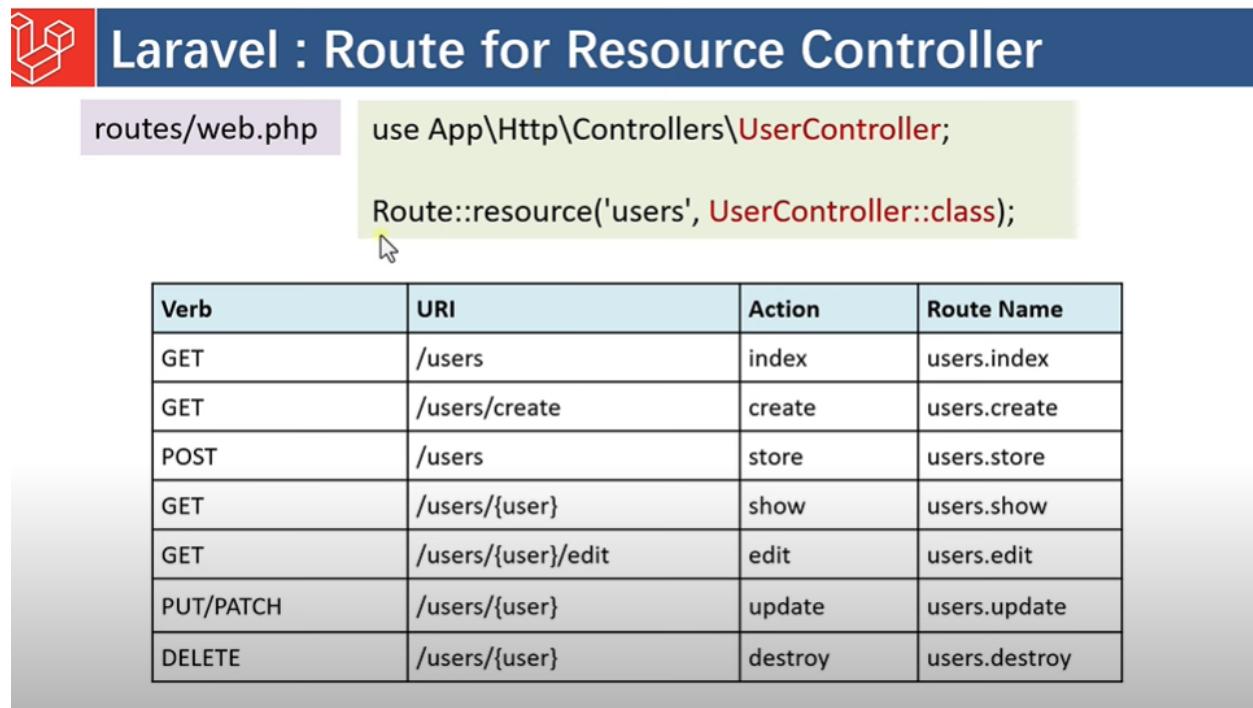
Example



Laravel : Resource Controller

```
UserController.php
class UserController extends Controller
{
    public function index(){}
    public function create(){}
    public function store(Request $request){}
    public function show(User $user){}
    public function edit(User $User){}
    public function update(Request $request, User $user){}
    public function destroy(User $user){}
}
```

The screenshot shows a code editor window for a Laravel application. The title bar says "Laravel : Resource Controller". The file being edited is "UserController.php". The code defines a UserController class that extends the Controller class. It contains seven methods: index, create, store, show, edit, update, and destroy. The "store" method takes a Request object as a parameter, and the "update" method takes both Request and User objects.



Laravel : Route for Resource Controller

```
routes/web.php
use App\Http\Controllers\UserController;

Route::resource('users', UserController::class);
```

Verb	URI	Action	Route Name
GET	/users	index	users.index
GET	/users/create	create	users.create
POST	/users	store	users.store
GET	/users/{user}	show	users.show
GET	/users/{user}/edit	edit	users.edit
PUT/PATCH	/users/{user}	update	users.update
DELETE	/users/{user}	destroy	users.destroy

The screenshot shows a code editor window for a Laravel application. The title bar says "Laravel : Route for Resource Controller". The file being edited is "routes/web.php". It contains a single line of code: "Route::resource('users', UserController::class);". Below the code editor is a table mapping HTTP verbs to their corresponding routes and actions. The table has four columns: Verb, URI, Action, and Route Name. The rows correspond to the methods defined in the UserController: GET /users (index), GET /users/create (create), POST /users (store), GET /users/{user} (show), GET /users/{user}/edit (edit), PUT/PATCH /users/{user} (update), and DELETE /users/{user} (destroy).

Only method show create update show

```
Route::resource('users', UserController::class)->only([
    'create', 'update', 'show'
]);
```

All method show without update and show method

```
Route::resource('users', UserController::class)->except([
    'update', 'show'
]);
```

Multiple Resource controller route

Laravel : Route for Multiple Resource Controller i

Single Resource Controller Route

```
Route::resource('users', UserController::class);
```

Multiple Resource Controller Route

```
Route::resource([
    'users' => UserController::class,
    'post' => PostController::class
]);
```

Nested Resource



Laravel : Nested Resources

i

routes/web.php

```
use App\Http\Controllers\UserCommentController;
```

Users → Comments

```
Route::resource('users.comments', UserCommentController ::class);
```

Verb	URI	Action	Route Name
GET	/users/{user}/comments	index	users.comments.index
GET	/users/{user}/comments/create	create	users.comments.create
POST	/users/{user}/comments	store	users.comments.store
GET	/users/{user}/comments/{comment}	show	users.comments.show
GET	/users/{user}/comments/{comment}/edit	edit	users.comments.edit
PUT/PATCH	/users/{user}/comments/{comment}	update	users.comments.update
DELETE	/users/{user}/comments/{comment}	destroy	users.comments.destroy

Yahoo
Baba

Nested resource

```
Route::resource('users', UserController::class);
```

```
Route::resource('users.comments', CommentController::class);
```

Eloquent



Laravel : Database Coding

i

Database Coding

Query Builder

```
DB::table('users')->all()
```

```
User::all()
```

Eloquent ORM

Eloquent : Clear & Strong

(object-relational mapper)

✓ Easy Syntax

✓ Complex Relationship between Tables

- one-to-one
- one-to-many
- many-to-many

✓ Observers and Scope

Yahoo
Baba



Laravel : Steps to work in Eloquent ORM

1 `php artisan make:model user --controller`

```
use App\Models\User;

class UserController extends Controller
{
    public function show()
    {
        $users = User::all();
        return $users;
    }
}
```

2 `use App\Http\Controllers\UserController;`

```
Route::get('/user', [UserController::class, 'show']);
```

- Read – get()
- Insert – create()
- Update – update()
- Delete – delete()

<http://localhost/user>



Laravel : Join Tables with Eloquent ORM

- INNER JOIN `join()`
- LEFT JOIN `leftJoin()`
- RIGHT JOIN `rightJoin()`
- CROSS JOIN `crossJoin()`



Laravel : Read Data with Eloquent ORM

```
SELECT * FROM users
```

```
User::all()
```

```
SELECT name, city FROM users
```

```
User::select('name', 'city')->get()
```

```
SELECT * FROM users WHERE city = 'goa'
```

```
User::where('city', '=', 'goa')->get()
```

```
User::where('city', '=', 'goa') ->where('age', '>', 18)->get()
```

```
User::where('city', '=', 'goa') ->orWhere('age', '>', 18)->get()
```

- `whereNot()`
- `whereBetween()`
- `whereIn()`
- `whereNull()`
- `whereNotNull()`
- `whereMonth()`
- `whereDate()`
- `whereDay()`
- `whereYear()`
- `whereTime()`



Laravel : Create Data with Eloquent ORM

i

Method : I

```
use App\Models\User;  
  
$user = new User;  
  
$user->name = 'Yahoo Baba';  
$user->email = 'yahoobaba@email.com';  
  
$user->save();
```

Method : II (Mass Assignment)

```
use App\Models\User;  
  
User::create([  
    'name' => 'Yahoo Baba',  
    'email' => 'yahoobaba@email.com',  
]);  
  
Models/User.php  
protected $guarded = [];  
  
protected $fillable = ['name', 'email'];  
Yahoo  
Baba
```

Note guarded holo....je gulo data show korte chai na seigulo guarded bitore rakhobo

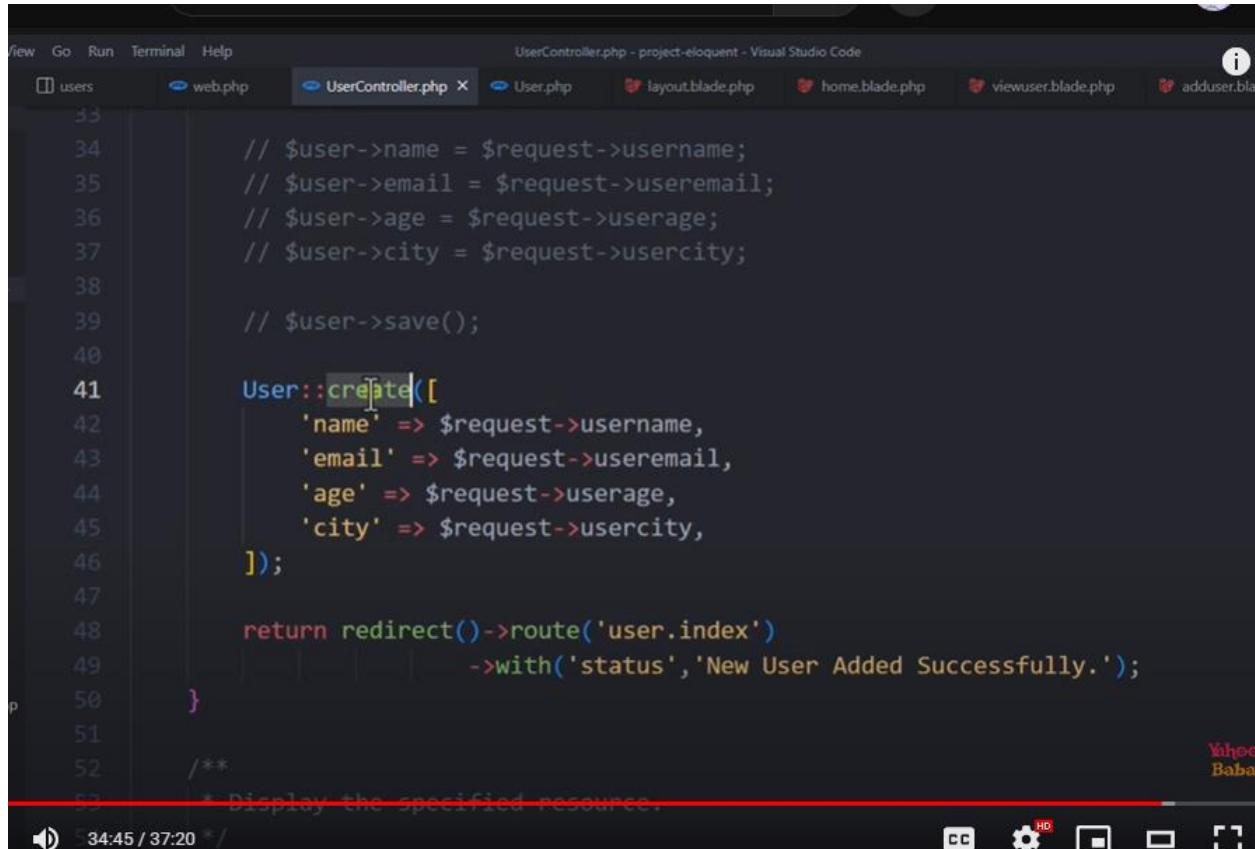
Fillable je gulo show korbo fillable bitore rakhobo

Create data

```
public function store(Request $request)  
{  
    $user = new User;  
  
    $user->name = $request->username;  
    $user->email = $request->useremail;  
    $user->age = $request->userage;  
    $user->city = $request->usercity;  
  
    $user->save();  
  
    return redirect()->route('user.index');  
}
```

Second method

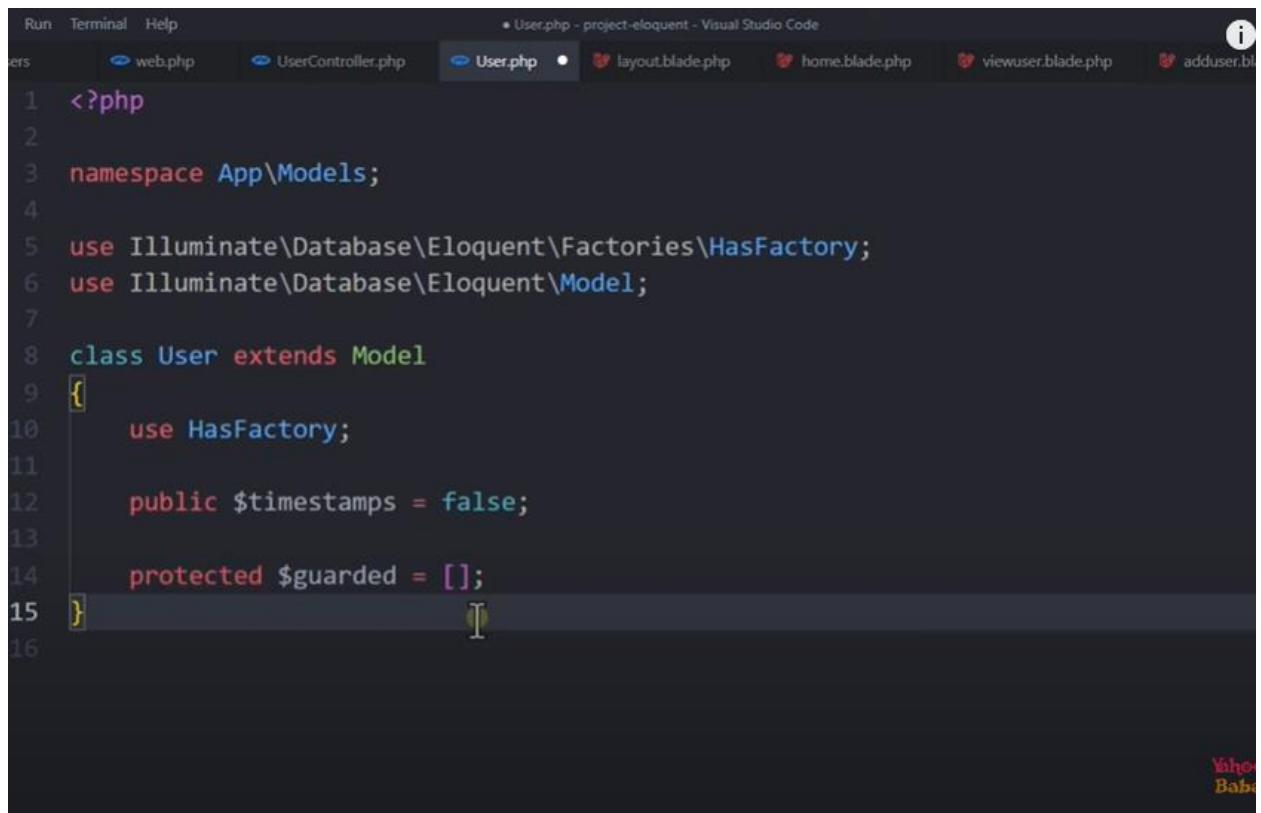
Step-1 controller



A screenshot of the Visual Studio Code interface showing the `UserController.php` file. The code implements a POST request handler for user creation:

```
view Go Run Terminal Help UserController.php - project-eloquent - Visual Studio Code
users web.php UserController.php X User.php layout.blade.php home.blade.php viewuser.blade.php adduser.blade.php
33
34     // $user->name = $request->username;
35     // $user->email = $request->useremail;
36     // $user->age = $request->userage;
37     // $user->city = $request->usercity;
38
39     // $user->save();
40
41     User::create([
42         'name' => $request->username,
43         'email' => $request->useremail,
44         'age' => $request->userage,
45         'city' => $request->usercity,
46     ]);
47
48     return redirect()->route('user.index')
49             ->with('status', 'New User Added Successfully.');
50 }
51
52 /**
53 * Display the specified resource.
54 */
34:45 / 37:20 /
```

Step-2 model

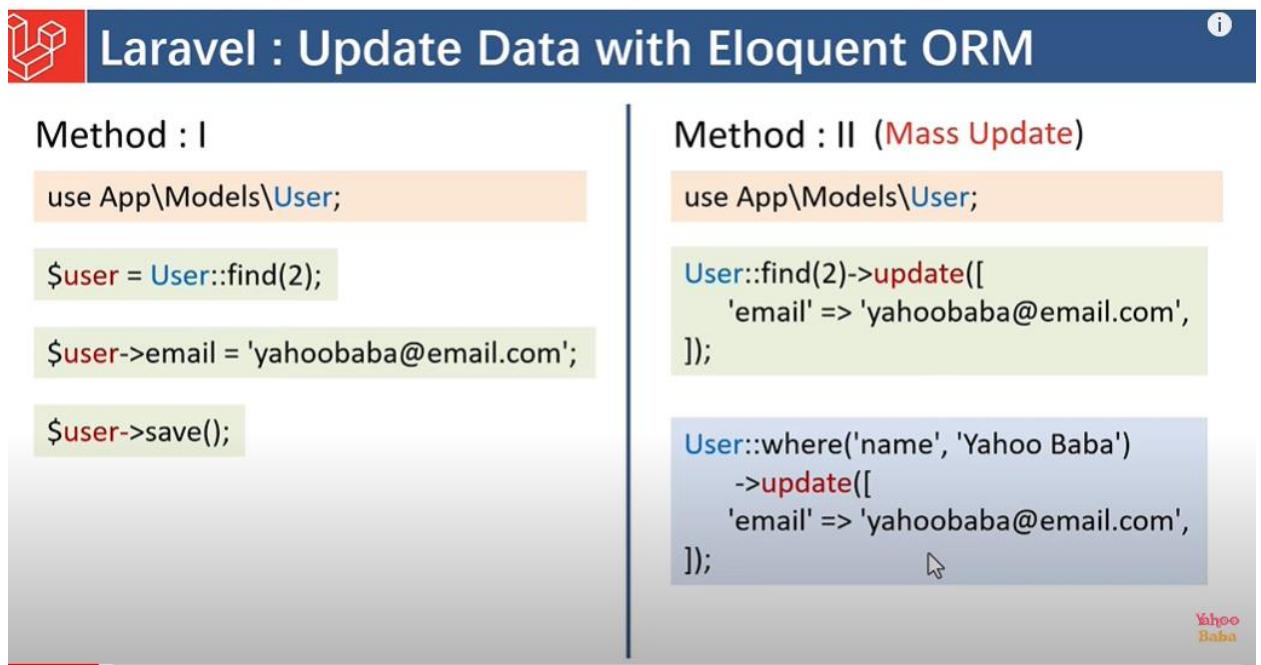


```
Run Terminal Help • User.php - project-eloquent - Visual Studio Code
users web.php UserController.php User.php ● layout.blade.php home.blade.php viewuser.blade.php adduser.blade.php

1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Database\Eloquent\Model;
7
8 class User extends Model
9 {
10     use HasFactory;
11
12     public $timestamps = false;
13
14     protected $guarded = [];
15 }
16
```

Update

Laravel : Update Data with Eloquent ORM



Method : I

```
use App\Models\User;

$user = User::find(2);

$user->email = 'yahoobaba@email.com';

$user->save();
```

Method : II (Mass Update)

```
use App\Models\User;

User::find(2)->update([
    'email' => 'yahoobaba@email.com',
]);
```

```
User::where('name', 'Yahoo Baba')
    ->update([
        'email' => 'yahoobaba@email.com',
    ]);

```



Laravel : Delete Data with Eloquent ORM

i

Method : I

```
use App\Models\User;  
  
$user = User::find(2);  
  
$user->delete();
```

Method : III

```
User::truncate();
```

- Delete all data in database table.
- Reset the auto-incrementing ID

Method : II (Mass Delete with IDs)

```
use App\Models\User;  
  
User::destroy(1);  
  
User::destroy(1, 2, 3);  
  
User::destroy([1, 2, 3]);
```

Yahoo
Baba

```
<td><a href="{{ route('user.show', $user->id) }}>cl</td>
```

Relationships

Step-1 foreign key migration file



Laravel : One To One Relationship

i

Students Table

ID	Name	Age
1	Ram Kumar	19
2	Salman Khan	18
3	Meera Khan	19
4	Sarita Kumari	21

PRIMARY KEY

Contacts Table

ID	Phone	City	Student_id
1	9988441122	Agra	1
2	8833554477	Mumbai	2
3	2233665544	Delhi	3
4	1122334455	Mumbai	4

FOREIGN KEY

```
$table->foreign('Student_id')->references('id')->on('students');
```

Yahoo
Baba

Step-2:model

Laravel : One To One Relationship

Students Table			Contacts Table			
id	Name	Age	id	Phone	City	Student_id
1	Ram Kumar	19	1	9988441122	Agra	1
2	Salman Khan	18	2	8833554477	Mumbai	2
3	Meera Khan	19	3	2233665544	Delhi	3
4	Sarita Kumari	21	4	1122334455	Mumbai	4

① App/Models/Student.php → App/Models/Contact.php

```
public function contact(){  
    return $this->hasOne(Contact::class);  
}
```

②

Laravel : hasOne() Method

Students Table			Contacts Table			
id	Name	Age	id	Phone	City	Student_id
1	Ram Kumar	19	1	9988441122	Agra	1
2	Salman Khan	18	2	8833554477	Mumbai	2
3	Meera Khan	19	3	2233665544	Delhi	3
4	Sarita Kumari	21	4	1122334455	Mumbai	4

```
return $this->hasOne(Contact::class, 'foreign_key', 'local_key');
```

↓ ↓

Student_id id

Step-3



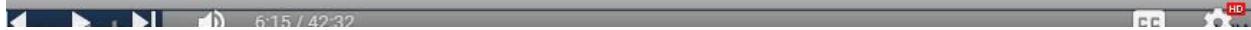
Laravel : One To One Relationship

3 StudentController.php

```
use App\Models\Student;
use App\Models>Contact;

class StudentController extends Controller
{
    public function show()
    {
        $students = Student::with('contact')->get();
        return $students;
    }
}
```

Function Name



Show the contact table all item

```
y-print []
{
    "id": 1,
    "name": "Yahoo Baba",
    "age": "20",
    "gender": "M",
    "contact": {
        "id": 1,
        "email": "yahoobaba@email.com",
        "phone": "123456789",
        "address": "#123 YB Road",
        "city": "Chandigarh",
        "student_id": 1
    },
    "id": 2,
    "name": "Salman Khan",
    "age": "22",
    "gender": "M",
    "contact": {
        "id": 2,
        "email": "salman@email.com",
        "phone": "3344556677",
        "address": "#345, SK Road",
        "city": "Mumbai",
        "student_id": 2
    }
}
```

Yahoo
Baba

Inverse Relationship



Laravel : Inverse Relationship

i

Students Table

id	Name	Age
1	Ram Kumar	19
2	Salman Khan	18
3	Meera Khan	19
4	Sarita Kumari	21

Contacts Table

id	Phone	City	Student_id
1	9988441122	Agra	1
2	8833554477	Mumbai	2
3	2233665544	Delhi	3
4	1122334455	Mumbai	4

1

App/Models/Student.php

2

App/Models/Contact.php

```
public function student(){
```

```
    return $this->belongsTo(Student::class);
```

```
}
```

One to Many relationship



Laravel : One To Many Relationship

i

Students Table

id	Name	Age
1	Ram Kumar	19
2	Salman Khan	18
3	Meera Khan	19
4	Sarita Kumari	21

PRIMARY KEY

id	Book	Date	Student_id
1	ABC	01-01-2024	1
2	XYZ	01-01-2024	1
3	MNC	02-01-2024	3
4	SRT	03-01-2024	3

FOREIGN KEY

Step-1 foreign key setup

```
$table->foreign('Student_id')->references('id')->on('students');
```

Step 2 :create function in model file

Laravel : One To Many Relationship

Students Table			Books Table			
id	Name	Age	id	Book	Date	Student_id
1	Ram Kumar	19	1	ABC	01-01-2024	1
2	Salman Khan	18	2	XYZ	01-01-2024	1
3	Meera Khan	19	3	MNC	02-01-2024	3
4	Sarita Kumari	21	4	SRT	03-01-2024	3

① App/Models/Student.php → App/Models/Book.php

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

Step-3



Laravel : One To Many Relationship

3 StudentController.php

```
use App\Models\Student;
use App\Models\Book;

class StudentController extends Controller
{
    public function show()
    {
        $students = Student::with('book')->get();
        return $students;
    }
}
```

Many to Many Relationship



Laravel : Many To Many Relationship

Users Table

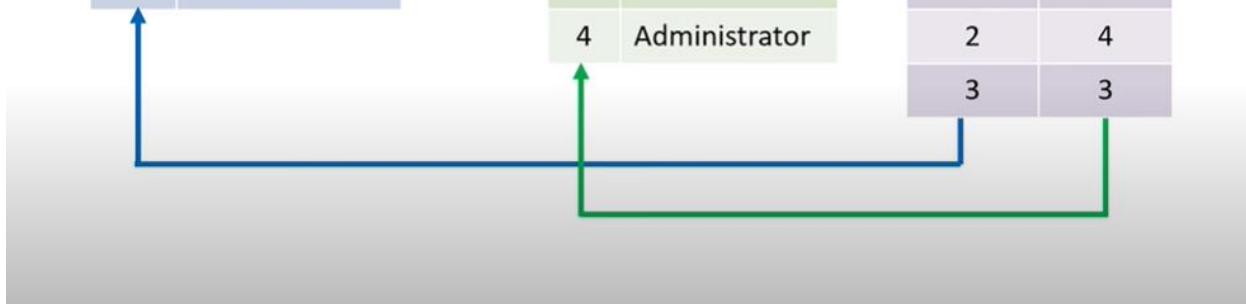
id	Name
1	Ram Kumar
2	Salman Khan
3	Sarita Kumari

Roles Table

id	Roles
1	Author
2	Editor
3	Contributor
4	Administrator

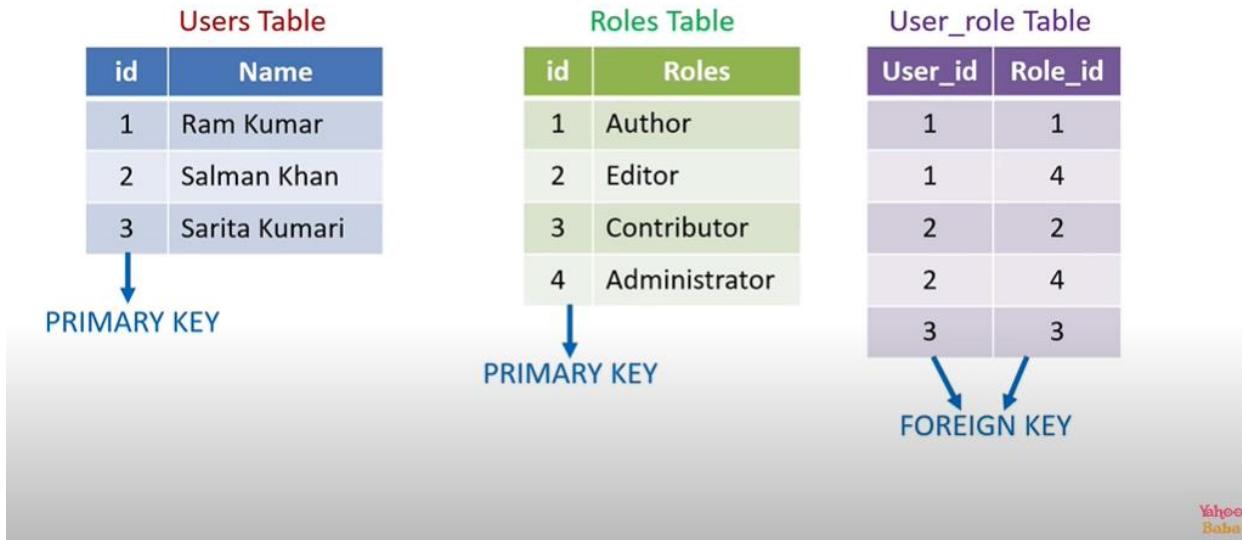
User_role Table

User_id	Role_id
1	1
1	4
2	2
2	4
3	3

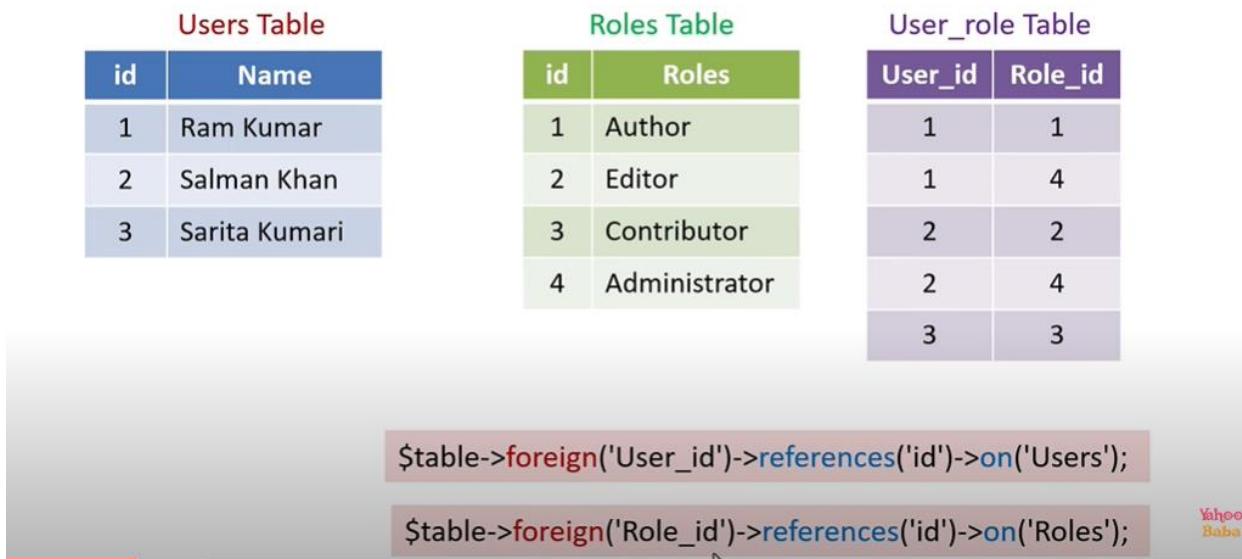




Laravel : Many To Many Relationship



Laravel : Many To Many Relationship





Laravel : Many To Many Relationship

Users Table

ID	Name
1	Ram Kumar
2	Salman Khan
3	Sarita Kumari

Roles Table

ID	Roles
1	Author
2	Editor
3	Contributor
4	Administrator

User_role Table

User_id	Role_id
1	1
1	4
2	2
2	4
3	3

1 Models/User.php

```
public function roles(){
    return $this->belongsToMany (Role::class, 'user_role');
```

Models/Role.php

Models/User_role.php

Yahoo
Baba



Laravel : Many To Many Relationship

3 UserController.php

```
use App\Models\User;
use App\Models\Role;

class UserController extends Controller
{
    public function show()
    {
        $user = User::find(1);
        return $user->roles;
    }
}
```

Laravel Eloquent with json data

YouTube BD

Search

L 9+

Laravel : JSON Column Data

Users Table

id	meta_data
1	<pre>{ 'name' : 'Yahoo Baba', 'email' : 'yahoobaba@email.com', 'mobile_number' : '11223344' }</pre>
2	<pre>{ 'name' : 'Amitabh Bachchan', 'email' : 'amitabh@email.com', 'mobile_number' : '22334455' }</pre>

1 \$table->json('meta_data')->nullable();

2 Models/User.php

```
protected $casts = [
    'meta_data' => 'json',
];
```

Edit Selection View Go Run Terminal Help 2024_04_09_073424_create_test_table.php - json-data - Visual Studio Code

```

6
7     return new class extends Migration
8     {
9         /**
10          * Run the migrations.
11         */
12         public function up(): void
13         {
14             Schema::create('tests', function (Blueprint $table) {
15                 $table->id();
16                 $table->json("meta_data")->nullable();
17                 $table->timestamps();
18             });
19         }
20
21         /**
22          * Reverse the migrations.
23          */
24         public function down(): void
25         {
26             Schema::dropIfExists('test');
27         }
28     }

```



Laravel : JSON Column Data

Users Table

	id	meta_data
	1	{ 'name' : 'Yahoo Baba', 'email' : 'yahoobaba@email.com', 'mobile_number' : '11223344' }
	2	{ 'name' : 'Amitabh Bachchan', 'email' : 'amitabh@email.com', 'mobile_number' : '22334455' }

4

UserController.php

```
use App\Models\User;  
  
class UserController extends Controller  
{  
    public function show()  
    {  
        $user = User::find(1);  
        return $user->meta_data;  
    }  
}
```

Yahoo
Baba

Terminal Help • TestController.php - json-data - Visual Studio Code

```
TestController.php ●  
  
// $test->meta_data = [  
//     'name' => 'Yahoo Baba',  
//     'email' => 'yahoobaba@email.com',  
//     'mobile_number' => '11223344'  
// ];  
// $test->save();  
  
$test = Test::create([  
    'meta_data' => [  
        'name' => 'Katrina Kaif',  
        'email' => 'katrina@email.com',  
        'mobile_number' => '88334455',  
        'a'  
    ]  
]);  
}  
  
/**
```

Model Event

Laravel : Model Events

CRUD

- Create → Send Email Notification to all Subscribers.
- Read → Increment the Read counter.
- Update → Send Email Notification to website Administrator.
- Delete → When a user's account is deleted, all posts related to that user get deleted.

Yahoo Baba

Laravel : Model Events

CRUD

Events :

- Create
 - **Creating**: before a record has been created.
 - **Created**: after a record has been created.
- Read
 - **Saving**: before a record is saved.
 - **Saved**: after a record has been saved.
- Update
 - **Updating**: before a record is updated.
 - **Updated**: after a record has been updated.
- Delete
 - **Deleting**: before a record is deleted or soft-deleted.
 - **Deleted**: after a record has been deleted or soft-deleted.
 - **Restoring**: before a soft-deleted record is going to be restored.
 - **Restored**: after a soft-deleted record has been restored.
 - **Retrieved**: after a record has been retrieved.

Yahoo Baba



Laravel : Database Structure

i

Users Table

id	Name
1	John Abraham
2	Salman Khan
4	Katrina Kaif

Posts Table

id	Title	User_id
1	Post One	2
2	Post Two	1
3	Post Three	3
4	Post Four	1
5	Post Five	2
6	Post Six	1

- Model Events
- Observers

Comments Table

id	detail	post_id
1	Good News	2
2	I Like it	1
3	Bad news	1
4	Wow Great	2
5	Ok good	3
6	When ?	3
7	I don't like it	5
8	wow....	4
9	Text....	5

Yahoo
Baba

Note :users table ekta name delete korle jotogulo post table and comments table er sokol kisu delete hoye jabe

Model event :model event using only small event handling



Laravel : Model Events

1 App/Models/Post.php

2 protected static function booted(): void {
 static::created(function(\$post) {
 //
 });

 static::deleted(function(\$post) {
 //
 });
}

- Creating
- Created
- Saving
- Saved
- Updating
- Updated
- Deleting
- Deleted
- Restoring
- Restored
- ➔ Retrieved

Normal rule : controller working

```
/*
public function create()
{
    $user = User::find(1)->delete();
    Post::where('user_id', 1)->delete();
}

/**
```

Step-1

```
File Edit Selection View Go Run Terminal Help
User.php - model-events - Visual Studio Code
EXPLORER ... User.php UserController.php
MODEL-EVENTS
app
Http\Controllers
Controller.php
PostController.p...
UserController.p...
Models
Post.php
User.php
Providers
bootstrap
config
database
public
resources
routes
console.php
web.php
storage
tests
vendor
editorconfig
.env
.env.example
.gitattributes
.gitignore
artisan
composer.json
6 use Illuminate\Database\Eloquent\Model;
7
8 class User extends Model
9 {
10     use HasFactory;
11
12     public function post(){
13         return $this->hasMany(Post::class);
14     }
15
16     protected static function booted(): void{
17         static::deleted(function($user){
18             $user->post()->delete();
19         });
20     }
21 }
22
```

Step-2:

```
public function create()
{
    $user = User::find([1])->delete();
}
```

Laravel Observers

Observer handling using large event handling



Laravel : Observers



1 php artisan make:observer PostObserver --model=Post

2 namespace App\Observers;
use App\Models\Post;
class PostObserver{

```
    public function created(Post $post): void {
```

```
        };
```

```
    public function deleted(Post $post): void {
```

```
        };
```

```
}
```

app/Observers/PostObserver.php

Yahoo
Baba



Laravel : Observers



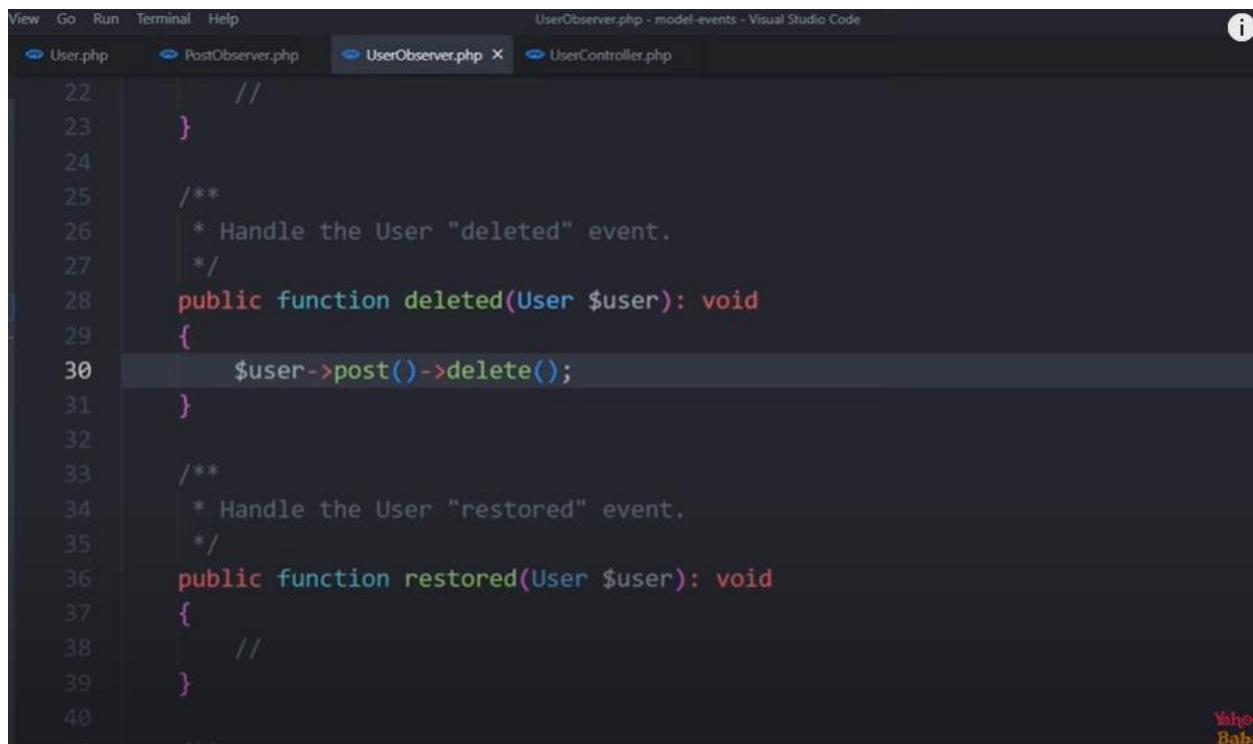
3 Providers/AppServiceProvider.php

```
use App\Models\Post;  
use App\Observers\PostObserver;  
  
class AppServiceProvider extends ServiceProvider  
{  
    public function boot(): void{  
        Post::observe(PostObserver::class);  
    }  
}
```

Yahoo
Baba

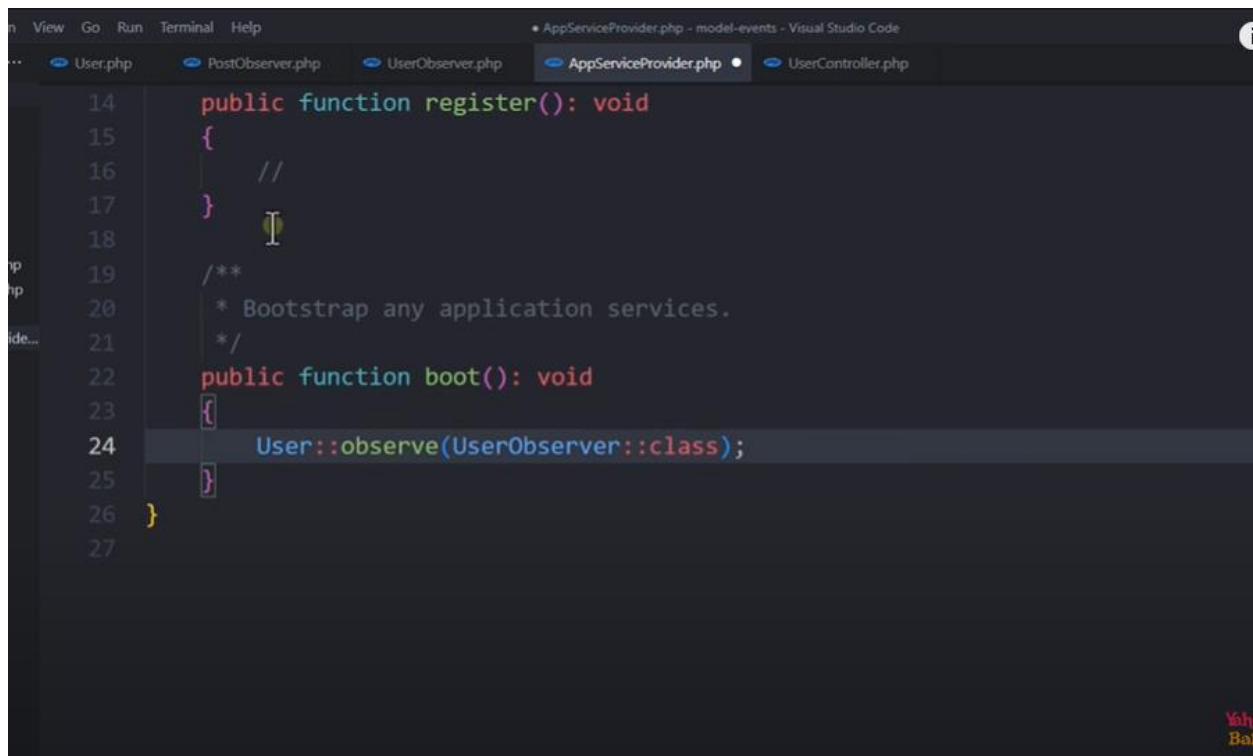
Step 1: create observer file

Step 2:



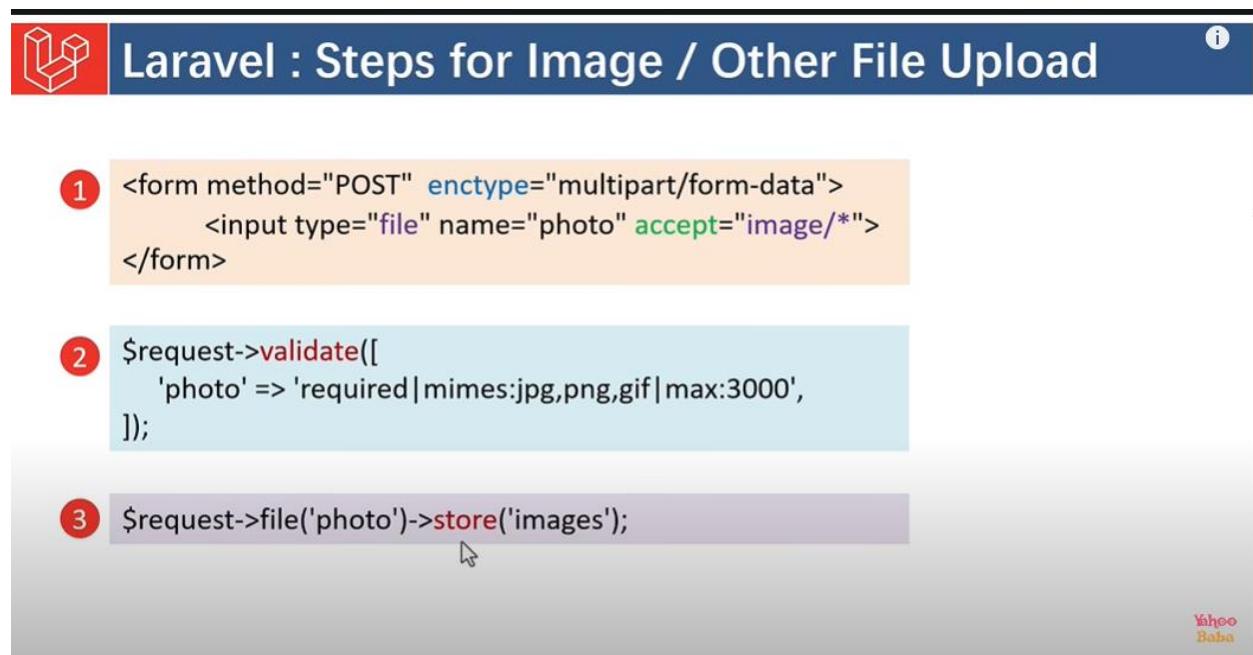
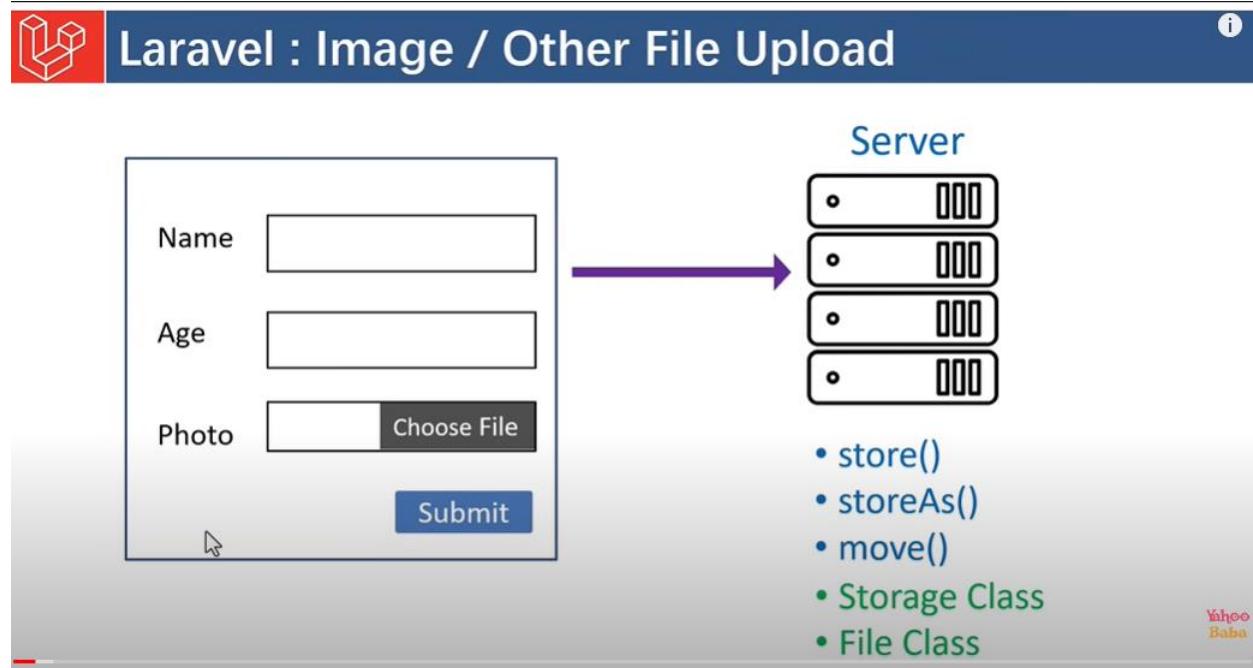
```
View Go Run Terminal Help UserObserver.php - model-events - Visual Studio Code
User.php PostObserver.php UserObserver.php ✘ UserController.php
22     //
23 }
24
25 /**
26 * Handle the User "deleted" event.
27 */
28 public function deleted(User $user): void
29 {
30     $user->post()->delete();
31 }
32
33 /**
34 * Handle the User "restored" event.
35 */
36 public function restored(User $user): void
37 {
38     //
39 }
40
```

Step-3:



```
n View Go Run Terminal Help AppServiceProvider.php - model-events - Visual Studio Code
... User.php PostObserver.php UserObserver.php AppServiceProvider.php UserController.php
14     public function register(): void
15     {
16         //
17     }
18
19 /**
20 * Bootstrap any application services.
21 */
22 public function boot(): void
23 {
24     User::observe(UserObserver::class);
25 }
26
27
```

Laravel Image & File Upload



Controller

Database link add (all time using)

```

$request->validate([
    'photo' => 'required|mimes:png,jpg,jpeg|max:3000'
]);

$file = $request->file('photo');

$path = $request->photo->store('image','public');
// return $path;

User::create([
    'file[name' => $path,
]);
return redirect()->route('user.index')->with('status','User Image Uploaded Successfully.');

```

Without database link add

```

*/
public function store(Request $request)
{
    $file = $request->file('photo');

    $request->validate([
        'photo' => 'required|mimes:png,jpg,jpeg|max:3000'
    ]);

    $path = $request->file('photo')->store('image','public');

    return redirect()->route('user.index')->with('status','User Image Uploaded Successfully.');
}

/**

```

Other away

```

$file = $request->file('photo');

$request->validate([
    'photo' => 'required|mimes:png,jpg,jpeg|max:3000'
]);

// $path = $request->photo->store('image','public');
$fileName = time(). '_'. $file->getClientOriginalName();
$path = $request->photo->storeAs('image',$fileName,'public');
return $path;

return redirect()->route('user.index')->with('status','User Image Uploaded Successfully.');

```

Success status

```
<div class="row">
    <div class="col-6">
        @if (session('status'))
            <div class="alert alert-success">
                {{ session('status') }}
            </div>
        @endif
    </div>
</div>
```

Image read from database

```
@foreach ($users as $user)
    <div class="col-2">
        
    </div>
@endforeach
```

Delete image:

```
public function destroy(string $id)
{
    $user = User::find($id);

    $user->delete();

    $image_path = public_path("storage/") . $user->file_name;

    if(file_exists($image_path)){
        @unlink($image_path);
    }

}
```

```

<form action="{{ route('user.destroy', $user->id) }}" method="POST">
    @csrf
    @method('DELETE')
    <button type="submit" class="btn btn-danger btn-sm mb-3">Delete</button>
</form>

```

Update :

```

public function update(Request $request, string $id)
{
    $user = User::find($id);

    if($request->hasFile('photo')){

        $path = $request->photo->store('image', 'public');

        $user->file_name = $path;
        $user->save();

        return redirect()->route('user.index')
            ->with('status', 'User Image Updated Successfully.');
    }
}

```

```

public function update(Request $request, string $id)
{
    $user = User::find($id);

    if($request->hasFile('photo')){

        $image_path = public_path("storage/") . $user->file_name;

        if(file_exists($image_path)){
            @unlink($image_path);
        }

        $path = $request->photo->store('image', 'public');

        $user->file_name = $path;
        $user->save();

        return redirect()->route('user.index')
            ->with('status', 'User Image Updated Successfully.');
    }
}

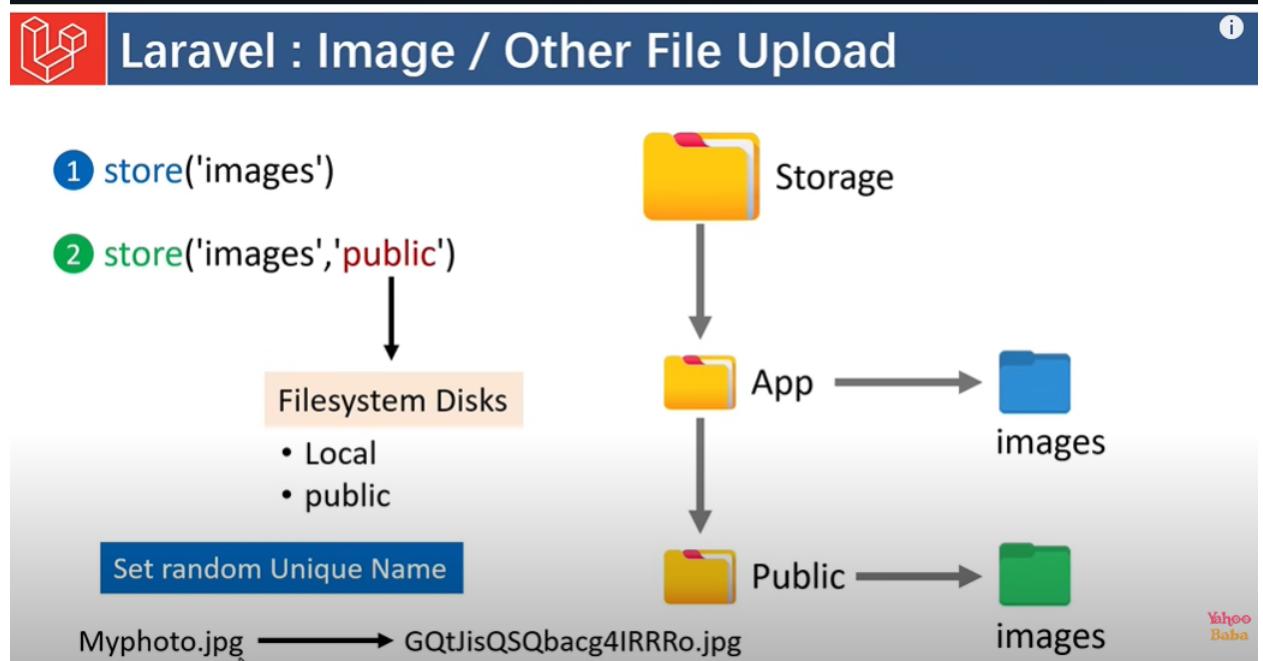
```

```

e Edit Selection View Go Run Terminal Help
file-update.blade.php - file-upload - Visual Studio Code
upload.blade.php UserController.php file-update.blade.php X web.php
7 <form action="{{ route('user.update', $user->id) }}" method="POST" enctype="multipart/form
8 @csrf
9 @method('PUT')
0 <div class="row">
1     <div class="col-3">
2         
3     </div>
4     <div class="col-9">
5         <input type="file" name="photo" accept=".jpg,.png,.jpeg">
6         @error('photo')
7             <div class="alert alert-danger mb-1 mt-1">{{ $message }}</div>
8         @enderror
9     </div>
0     <div class="col-12">
1         <input type="submit" class="btn btn-sm btn-primary">
2     </div>
3 </div>

class="col-9">
put type="file" onchange="document.querySelector('#output').src=window.URL.createObjectURL(this.files[0])" name="photo"
@error('photo')
<div class="alert alert-danger mb-1 mt-1">{{ $message }}</div>

```





Laravel : Image / Other File Upload

i

1 `store('images')`

2 `store('images','public')`

3 `storeAs('images', 'filename.jpg','public')`

4 `move(base_path("\myfolder\images"), 'filename.jpg');`

- `Public_path()`
- `Storage_path()`
- `Resource_path()`

Yahoo
Baba

Laravel Components

Static components

The screenshot shows a browser window with the YouTube BD homepage. Overlaid on it is a blue header bar with the text "Laravel : Steps to Create Components". Below this, two code snippets are displayed side-by-side.

Step 1: `php artisan make:component Alert`

Step 2: `app/View/Components/alert.php`

```
class Alert extends Component
{
    public function __construct()
    {
    }

    public function render()
    {
        return view('components.alert');
    }
}
```

Step 3: `resources/views/components/alert.blade.php`

```
<div class="alert alert-success alert-dismissible fade show" role="alert">
    Alert Message Goes Here.
</div>
```

Yahoo
Baba



Laravel : Steps to Create Components

Any View Blade File :

3

```
<html>
  <body>
    <x-alert/>
    <x-alert/>
    <x-alert/>
  </body>
</html>
```

Alert Message Goes Here

X

Alert Message Goes Here

X

Alert Message Goes Here

X

Yahoo
Baba

Note: x-component name dite hobe

Dynamic components

Step-1

```
<body>
  <x-alert type="success" message="This is success message alert." />
  <x-alert type="error" message="This is error message alert."/>
  <x-alert type="info" message="This is info message alert." />
</body>
</html>
```

Step-2

```
use Illuminate\View\Component;

class alert extends Component
{
    public $type;
    public $message;
    /**
     * Create a new component instance.
     */
    public function __construct(string $type, string $message)
    {
        $this->type = $type;
        $this->message = $message;
    }
}
```

Step-3



A screenshot of the Visual Studio Code interface. The title bar shows "alert.blade.php" as the active file. The code editor displays the following Blade template:

```
1 <div class="alert alert-{{ $type }}" role="alert">
2     {{ $message }}
3 </div>
```

Other

Passing data

```
<!-- Use the ProductList component and pass the products -->
<x-product-list :products="$products" />
```

View data

blade

 Copy code

```
<ul>
    @foreach ($products as $product)
        <li>{{ $product->name }} - ${{ $product->price }}</li>
    @endforeach
</ul>
```

Class components

```
class ProductList extends Component
{
    public $products;

    // Accept products as a parameter
    public function __construct($products)
    {
        $this->products = $products;
    }

    public function render()
    {
        // Render the Blade component view
        return view('components.product-list');
    }
}
```

Slots components

Note :html tag passing using slot components



Laravel : Slots



1 `php artisan make:component Alert`

Heading Goes here

x

Alert Message Goes Here

2 `<div class="alert alert-success alert-dismissible fade show" role="alert">
 {{ $slot }}
 <button type="button" class="btn-close" data-bs-dismiss="alert"></button>
</div>`

3 `<x-alert>
 <h4>Heading Goes Here</h4>
 <hr/>
 <p>Alert Message Goes Here</p>
</x-alert>`

Yahoo
Baba