**Laravel Installation and Configaration**

Download then setup to **xampp>php>php.exe**Laravel installation using **composer global require "laravel/installer"**

Now open command window using **ctrl+shift+right click to the htdocs folder** then paste the 2 no code.

Now we have to search laravel then ctrl+shift+right click and open command window and write **php artisan serve** for server start

**Routing & Controller in Laravel**

**Routing** হচ্ছে URL; কোন পেজ view করার জন্য কোন নাম define করে দেওয়াই হচ্ছে Routing এর কাজ। i.e. localhost/**contact**, localhost/**about** etc.

Routes.php page এ যে get method টা আছে এটি দুটি parameter নেয়; একটি **string** and **anonymous function** or **controller@method** and **View** থাকে হচ্ছে Resource folderএর ভিতর। **Controller** থাকে হচ্ছে App>HTTP>Controllers.

View এর প্রতিটি পেজকে আমরা Route থেকে Call করতে পারি কিন্তু এটি কোন সঠিক পদ্ধতি না; সেজন্য আমরা View এর পেজ সমূহকে Controller এর মাধ্যমে Call করব। Controller তৈরি করার জন্য Command হচ্ছে **php artisan make:controller page\_name**

প্রতিটি পেজে master page কে import বা inherit করে কাজ করতে হয় (এটি parent class এবং এক একটি page হচ্ছে object. Master পেজ create করতে হয় View এ।

**কিভাবে page, css, script, image, link etc. dynamic করতে হয় (using laravel 5)**:

**এজন্য ২টি কাজ করতে হয়**:

১. Page গুলোকে পোজেক্ট ফোল্ডারে রাখতে হয়

২. CSS ও অন্যান্য Assets গুলোকে Public ফোল্ডারে রাখতে হয়।

**তারপর**

HTML::script() is used for .js files and HTML::style() is used for css.

{{ Html::style( 'css/app.css') }}

[**N.B**, Default Composer package illuminate/html ]

অথবা, <link href="{{ asset('css/masterCss.css') }}" rel="stylesheet">

Or

<link href="/css/master.css" rel="stylesheet">

OR

{{ asset('css/app.css') }}

[**N.B**, Curly brackets and asset method are blade syntax]

{!! Html::style( asset('css/app.css')) !!}

[**N.B**, !! comes from composer package collective/html]

1. **Laravel Collective:**

In cmd run command:

composer require laravelcollective/html

In your composer.json file in "require" group should be added this line:

"laravelcollective/html": "5.2.\*"

Update composer with command:

**composer update**

In config/app.php add this lines:

in providers group:

Collective\Html\HtmlServiceProvider::class,

in aliases group:

'Form' => Collective\Html\FormFacade::class,

'Html' => Collective\Html\HtmlFacade::class,

Now we can dynamic our Link, CSS, Images, Script like this:

**{!! Html::style('assets/bootstrap/css/bootstrap.css')!!}**

**Making Database**

1. Database এর সাথে Connection:

Vendor>env এই env ফাইলটাই Database Connection ফাইল।

Database তৈরির Code:

**In command line:**

php artisan make:migration create\_users\_table --create=users

তাহলে user table তৈরি হবে।

Table এ ফিল্ড add করার পরে Run করতে হবে তা না হলে ফিল্ড add হবে না।

**Command Line:**

php artisan migrate

এবং Rollback বা আগের অবস্থায় ফিরে আসার জন্য বা এক স্টেপ পিছেনে আশার জন্য **Command line**:

php artisan migrate:rollback

1. **Mass Assignment** for getting data to **model** from controller for storing to database: Model তৈরির জন্য **Class** এ নিচের Code লিখতে হবে।

protected $table = '$users';

protected $primaryKey='$user\_id';

protected $fillable = ['id', 'password'];

**Form Making**

1. **Restful Resource Controller:**

Restful Resource Controller তৈরি করলে Route.php পেইজ এ Route::get এর পরিবর্তে Route::resource ব্যবহার করে Controller টিকে include করতে হবে যাতে সহজে Image বা অন্যান্য ফাইল নিয়ে কাজ করা যায় এবং ফাংশনটা হবে নিচের মত:

Route::resource('registration','dbController');//Only Controller Load হবে, কোন ফাংশন হবে না।

1. {{ Form::open(array('route' => 'registration.store')) }}

//Html Form

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

এটার মাধ্যমে URL এ ডাটা পাঠানোর জন্য **page\_name.store** ('registration.store')

অথবা নরমালভাবেও পাঠানো যায় এক্ষেত্রে ফর্ম এর নিচে {{Form::token()}} এটা দিতে হবে।

1. **Accessing The Request** or **Input request** (Form Data controller থেকে দেখা):

public function store(Request $request){

$input=$request->all();

return $input;

}

1. **Connection between Controller and Model (Controller to Model):**

public function store(Request $request)

{

$input=$request->all();

//return $input;

myFirstModel::create($input);

return redirect('success');

}

url:localhost:8000/registration/create

1. **Model to Controller (Retrieving Multiple Models):**

public function index()

{

$allDBdata = myFirstModel::all();

return $allDBdata;

}

url:localhost:8000/registration

**Pagination Creation**

1. **Pagination (Paginating Eloquent):**

public function index()

{

$allDBdata = myFirstModel::paginate(1); //all() **ফাংশনটি দেওয়ার দরকার নাই**

return $allDBdata;

}

1. **Displaying Results In A View (Controller to View):** compact or with **ফাংশনের মাধ্যমে পাঠাতে হবে**

public function index()

{

$allDBdata = myFirstModel::paginate(1);

return view('dbTest.index', compact('allDBdata'));

//return $allDBdata;

}

url:localhost:8000/registration

**CRUID Operaton**

1. **Insert, update, delete oparations:**

**Edit or Update (View to URL): URL এ আমরা ID পাঠালাম বা Using Link**

<td><a class="btn btn-info" href="{!! Route('reg.edit',$singleDBdata->user\_id) !!}">Edit</a></td>

**Query in Controller by ID and Controller to view:**

public function edit($id)

{

$allDBdata=myFirstModel::findOrFail($id);

return view('dbTest.edit',compact('allDBdata'));

}

**In View after Update and Data will be sent from View to Controller:**

{!! Form::open(array('route' => ['reg.update', $allDBdata->user\_id],'method'=>'PUT')) !!}

**In controller:**

public function update(Request $request, $id)

{

$inputData=$request->all();

$dbQueryData=myFirstModel::findOrfail($id);

$dbQueryData->update($inputData);

return redirect('reg');

}

**Delete:**

**In View:** Using Form

<td>

{!! Form::open(array('route' => ['reg.destroy', $singleDBdata->user\_id],'method'=>'delete')) !!}

{!! Form::hidden($singleDBdata->user\_id) !!} <!-- this line is not needed -->

{!! Form::submit('Delete', ['class'=>'btn btn-danger']) !!}

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

</td>

**In Controller:**

public function destroy($id)

{

$dbQueryData=myFirstModel::findOrfail($id);

$dbQueryData->delete($dbQueryData);

return redirect('reg');

}

1. Searching Data from Database:

**In View:**

{!! Form::open(['route' => 'reg.index','method' => 'GET']) !!}

{{ Form::text('user\_name')}}

{{ Form::submit('Search')}}

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

**In Controller:**

public function index(Request $request)

{

$count=1;

$allDBdata = myFirstModel::orderBy('user\_name');

$user\_name=$request->input('user\_name');

if(!empty($user\_name)){

$allDBdata->where('user\_name', 'like', '%'.$user\_name.'%');

}

$allDBdata = $allDBdata->paginate(3);

return view('dbTest.index', compact('allDBdata','count'));

//return $allDBdata;

}

**For Two search boxes:**

**In view:**

{!! Form::open(['route' => 'reg.index','method' => 'GET']) !!}

<lavel>User Full name: </lavel>{{ Form::text('user\_fullname')}}

<lavel>Username: </lavel>{{ Form::text('user\_name')}}

{{ Form::submit('Search')}}

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

**In Controller:**

public function index(Request $request)

{

$count=1;

$allDBdata = myFirstModel::orderBy('user\_name');

$user\_fullname=$request->input('user\_fullname');

$user\_name=$request->input('user\_name');

if(!empty($user\_fullname)){

$allDBdata->where('user\_fullname', 'like', '%'.$user\_fullname.'%');

}

if(!empty($user\_name)){

$allDBdata->where('user\_name', 'like', '%'.$user\_name.'%');

}

$allDBdata = $allDBdata->paginate(3);

return view('dbTest.index', compact('allDBdata','count'));

//return $allDBdata;

}

1. Authentication or Login/Logout System

return

1. How to Register & Use Service Providers

**Introduction**: Go ahead and look at the contents of the config/app.php file. You'll find an array entry that lists all the service providers that will be loaded during the bootstrapping of the Laravel application (with core aplication). If we want to load a dependency or service from service container (bucket) we need to load it via service provider. We can also create a custom service provider that will also be loaded automatically by laravel. So we need to inform laravel about the new service provider in config/app.php file at providers array.

**What Is a Service Container?**

In the simplest terms, we could say that the service container in Laravel is a box that holds various components, and they are served as needed throughout the application.

**For example:**

<?php

namespace App\Library\Services;

class DemoOne

{

public function doSomethingUseful()

{

return 'Output from DemoOne';

}

}

**What is a Service Provider?**

If the service container is something that allows you to define bindings and inject dependencies, then the service provider is the place where (register method) it happens that has already been discussed earlier in introduction.

**For example:**

<?php

namespace App\Providers;

use Illuminate\Support\ServiceProvider;

use App\Library\Services\DemoOne;

class EnvatoCustomServiceProvider extends ServiceProvider

{

public function boot()

{

}

public function register()

{

$this->app->bind('App\Library\Services\DemoOne', function ($app) {

return new DemoOne();

});

}

}

**How to Register Custom Service Provider**: So you've created your custom service provider. That's great! Next, you need to inform Laravel about your custom service provider (in the file config/app.php) so that it can load it along with other service providers during bootstrapping.

**For example:**

/\*

\* Application Service Providers...

\*/

App\Providers\AppServiceProvider::class,

App\Providers\AuthServiceProvider::class,

App\Providers\BroadcastServiceProvider::class,

App\Providers\EventServiceProvider::class,

App\Providers\RouteServiceProvider::class,

App\Providers\EnvatoCustomServiceProvider::class,

**Register Method**:

To start with, we'll go through the register method to understand how you could actually use it. Open the service provider file app/Providers/EnvatoCustomServiceProvider.php that was created earlier and replace the existing code with the following.

**For example:**

<?php

namespace App\Providers;

use Illuminate\Support\ServiceProvider;

use App\Library\Services\DemoOne;

class EnvatoCustomServiceProvider extends ServiceProvider

{

public function boot()

{

}

public function register()

{

$this->app->bind('App\Library\Services\DemoOne', function ($app) {

return new DemoOne();

});

}

}

**Boot Method:**

The boot method which you could use to extend the core Laravel functionality. In this method, you could access all the services that were registered using the register method of the service provider. In most cases, you want to register your event listeners in this method, which will be triggered when something happens.

**For Example**:

public function boot()

{

View::share('key', 'value');

}

public function boot()

{

Validator::extend('my\_custom\_validator', function ($attribute, $value, $parameters, $validator) {

// validation logic goes here...

});

}

public function boot()

{

View::composer(

'demo', 'App\Http\ViewComposers\DemoComposer'

);

}

public function boot()

{

parent::boot();

Route::model('user', App\User::class);

}

**Dependency injection (uses):**

Here's the code somewhere in your controller where the dependency will be injected.

**For example:**

<?php

namespace App\Http\Controllers;

use App\Http\Controllers\Controller;

use App\Library\Services\DemoOne;

class TestController extends Controller

{

public function index(DemoOne $customServiceInstance)

{

echo $customServiceInstance->doSomethingUseful();

}

}

**Eloquent ORM & Query Builder**

Eloquent ORM is slower than Query Builder but we can make relationship with Models and easier to build & less code on the other side the Query Builder is a little bit hard but more faster.

**Eloquent ORM**

**Insert**:

//Method 01

$article = new Article;

$article->user\_id = Auth::user()->id;

$article->content = $request->content;

$article->live = (boolean)$request->live;

$article->post\_on = $request->post\_on;

$article->save();

//Method 02

Article::create($request->all());

At Model:

protected $fillable = [

'user\_id', 'content', 'live', 'post\_on'

];

N.B, protected $guarded = ['']; or protected $fillable [];

public function setLiveAttribute($value){

$this->attributes['live'] = (boolean)($value);

}

//Method 03

Article::create([

'user\_id' => Auth::user()->id,

'content' =>$request->content,

'live' => $request->live,

'post\_on' =>$request->post\_on

]);

**Select**:

//All Records

Method 01:

$articles = Article::all();

return view('articles.index', compact('articles'));

Method 02:

$articles = Article::whereLive(1)->get();

$return articles;

//Paginated Records

$articles = Article::paginate(10);

return view('articles.index', compact('articles'));

//With soft deleted records

$articles = Article::withTrashed()->paginate(10);

return view('articles.index', compact('articles'));

//Only soft deleted records

$articles = Article::onlyTrashed()->paginate(10);

return view('articles.index', compact('articles'));

**Update**:

**Delete**:

//Soft deleting

$article = Article::findOfFail($id);

$article->delete();

return redirect('/articles');

[**N.B**, For soft deleting we need an extra field to our table called **deleted\_at.** Hence we have to go to the model and import the class like this: **use Illuminate\Database\Eloquent\SoftDeletes;** and finally we have to add an attribute like this: **protected $dates = ['post\_on','deleted\_at'];** And then we have to go to the migration file and add like this: **$table→softDeletes();** and we have to refresh like this: **php artisan migrate:refresh**]

//Without retrieving model

Article::destroy($id);

return redirect('/articles');

//For Multiple Deleting

Article::destroy([1,2,3,4]);

return redirect('/articles');

//Force deleting or Hard deleting

$article = Article::findOrFail($id);

$article->forceDelete();

return redirect('/articles');

**Query Builder:**

**Insert**:

Method 01:

DB::table('articles')->insert($request->all());

Method 02: All but except \_token field

DB::table('articles')->insert($request->except('\_token'));

Method 03:

DB::table('articles')→insert([

'user\_id' => Auth::user()->id,

'content' =>$request->content,

'live' => $request->live,

'post\_on' =>$request->post\_on

]);

[N.B. Query Builder don’t support accessor and mutator]

**Select**:

Method 01:

$articles = DB::table('articles')->get();

dd($article);

Method 02:

$articles = DB::table('articles')->whereLive(1)->get();

dd($article);

Method 03: Only one article return

$article = Article::table('articles')->whereLive(1)->first();

dd($article);

**Update**:

**Delete**:

**Eloquent Accessors & Mutators OR Custom Attributes**

The accessor and mutator both are public function in Laravel model for getting and setting model's attributes.

An accessor will automatically be called by Eloquent when attempting to retrieve the value of the first\_name attribute:

$fullName = $user->full\_name;

It's for customizing a model's attributes or adding fake attributes.

On the other hand mutator is for setting a real attribute of a model.

Mutator will be automatically called when we attempt to set the value of the an attribute.

**Rename table name and add new fields**

php artisan make:migration rename\_posts\_to\_articles

**At up:**

Schema::rename('posts','articles');

**At down:**

Schema::rename('articles','posts');

**For adding new fields or modifying:**

php artisan make:migration add\_live\_and\_post\_on\_fields\_to\_the\_articles\_table --table=articles

**At up:**

$table->boolean('live')->default(0);

$tabli->timestamp('post\_on')->nullable;

**At down:**

$table->dropColumn(['live', 'post\_on']);

**Model in Laravel**

The Eloquent ORM included with Laravel provides a beautiful, simple ActiveRecord implementation for working with your database. Each database table has a corresponding "Model" which is used to interact with that table. Models allow you to query for data in your tables, as well as insert new records into the table.

**N.B,** If table’s name is Articles (plural) then we should create a model named Article (singular) then the model will communicate automatically if other then we have to inform the name of the table in the model. It will also help in model relationship.

//