Laravel

Composer

Composer is PHP dependency manager.

Use with php frameworks to resolve some dependent file regarding to your project.

Get required packages from composer.json file and install in vendor folder of framework.

Example

Composer required component-name

introductoin

- Laravel is advance framework of PHP.
- Used for develop web applications and API
- Laravel provides advance concepts that’s why we develop code very fast
- Laravel is most powerful and advance framework of PHP.

Framework

- Framework is programming tool for developing code very fast with less coding and error free code.

Some features

- Routing
- Database connection
- Well managed code
- Ready made functions for session
- Helpers and common functions
- Role based structure
- And many more.

- Learn Laravel is very easy of you have some basic idea of core php.

History and versions

- Current Laravel version is version 9.
- Laravel completely written in PHP.
- First version introduced in 2011.
- Developed by “Tylor otwell”
- Laravel is free and open source framework.

- Laravel have large developer community
- Laravel is very fast and simple
- Provides regular updates
- Awesome command line features.
- Every year Laravel provides major update.
- Bug fixing and minor updates release weekly and monthly basis.

Comparison

Laravel

- Run on Server
- Back end Framework
- Direct Connect with database
- Cant make single page application

React, Angular

- Run on Browsers.
- Front end tools.
- Needs API to get data from backend technology
- Use for create single page applications.

Major topics

- Installation
- MVC
- Basics
- Controller
- Routing
- Modal
- View
- Database
- API

Some important topics before continue

Framework

- Framework is a platform for developing some software application, web application and etc.
- **“a framework is a real or conceptual structure intended to serve as a support or guide for the building of something that expands the structure into something useful.”**

- Framework provides a common structure to develop application that's why you don't want to write code from scratch and you can develop your application in very less time with fully managed and well formatted code.

- You need very hard efforts to write robust code in core languages like php and other core languages. Now with framework you just need to learn concept of framework and you can apply it in many projects and applications.

Start with framework

- Defining a structure of project.
- Make functions for database
- Make functions for routing.
- Validations common functions and etc.

- * now a framework provides all the thinks by default so we don’t want to focus on basic stuff, we just need to concentrate on our application logic rest of part is completely handle by framework.

How to learn framework

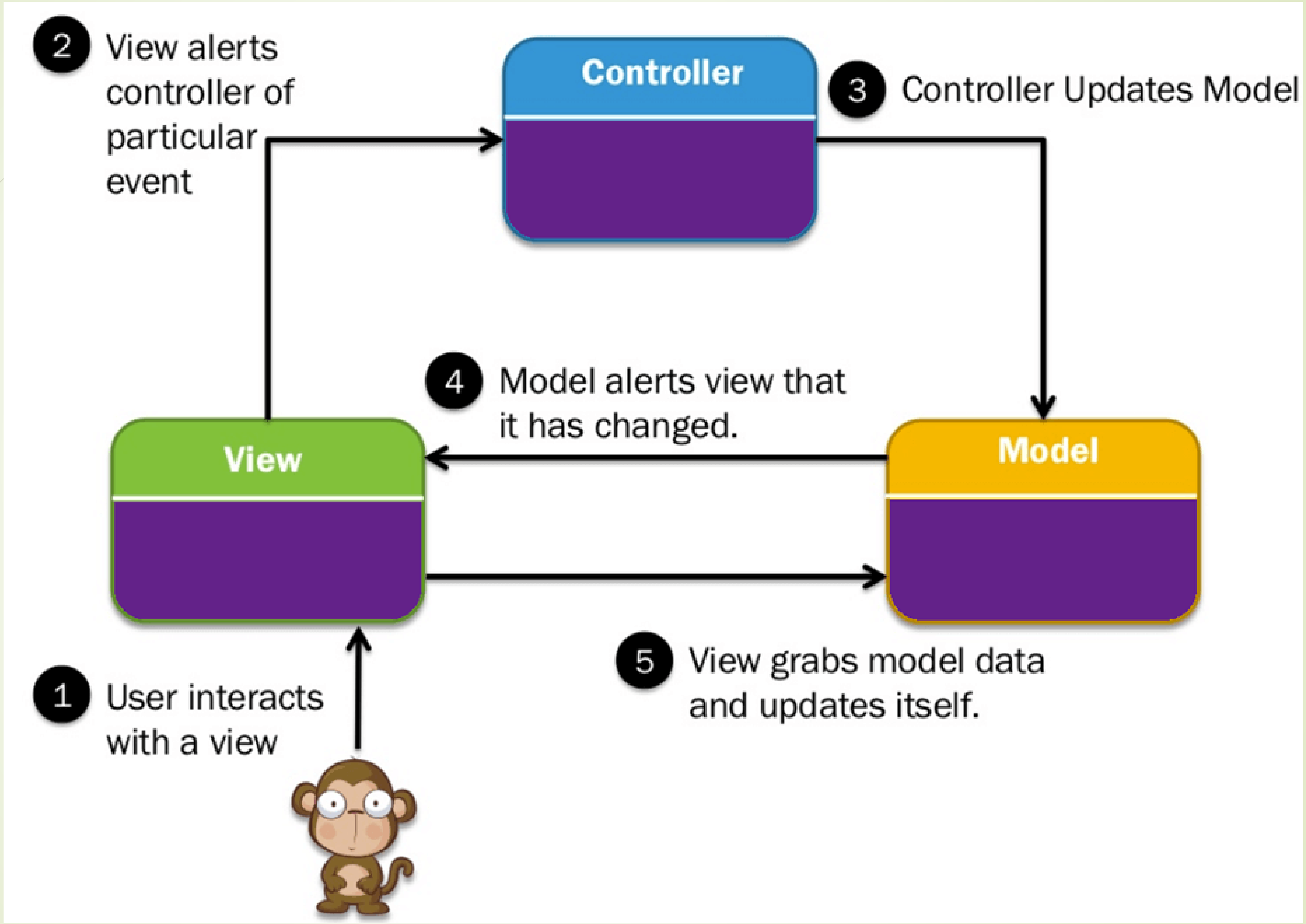
- First you need to learn core php language for Laravel.
- Do not try to remember code try to understand code.
- Do not deep in starting.
- Learn basics and make some hand on practice on it.
- Knowledge of OOP is very big plus point while learning Laravel
- Try some SQL query.
- Must install Xampp and composer
- Choose your preferred text editor like VS code or Sublime text.

Steps to learn laravel

- Learn how to install tools with command prompt
- Understand framework file structure.
- Learn basics and do some practice
- Step by step move on advance concepts.

MVC Architecture

- MVC (modal, View, Controller)
- Laravel follows MVC pattern / architecture to develop web application.
- You can divide your entire application in 3 parts
  - Modal : Data, Logic of Application.
  - View : user interface UI
  - Controller : route request to modal or view


2 View alerts controller of particular event
Controller
3 Controller Updates Model
4 Model alerts view that it has changed.
View
Model
5 View grabs model data and updates itself.
1 User interacts with a view

File example

- View for display data HTML code
- Controller get request from user and control modal and view
- Modal handle data structure, database and business logic

Laravel installation

- You can install Laravel in two ways.

1. Laravel installer
2. With composer

- You have php version 7.3 or greater for Laravel.
- You need composer

- Laravel Installer.

Composer global require Laravel/installer

* If you install Laravel via Laravel installer you never install Laravel again and again. Every time you create new project and done your tasks.

- Composer create-project –prefer-dist Laravel/Laravel project name
- This is method is very slow and not preferable

- After installation move to destination folder and create new project with following command

- Laravel new project_name

File and folder structure

App folder.

- Kernel.php
- Used for store commands.

- Exception
- Handler.php
- Used for exception handling codes

http

- All the controller and middleware stored in http folder
- Kernel used to register middle wares.

Modal folder

- Used for database tables management.

providers

- Used for authentication, routing and many more files about service providers.

Bootstrap folder

- Used to load Laravel code with HTML.

Cache folder

- Used to store cache files.

Config folder

- Used to store project configuration like app, authentication, database, cache. Cors, mail, session, login and etc.

Database

- Use to store important files about database.

Public folder

- Contains index file and other static files (.htaccess)

Resources

- Resources folder is used to store uncompiled files like css, js files and etc.

Views

- Used to store blade template files.
- HTML codes.

Routes

- All application routes are stored in routes folder.

Package.json

- Used to store application information

Edit first file in Laravel.

- Go to resources
- Views
- Edit welcome.blade file

Create new view

- Go to resources
- Go to views
- Create new file as filename.blade.php

*you need to edit route file for load newly created file

Web.php file set route to newly created file.

Routing in Laravel

- Routing in Laravel means navigate another section via URL is Routing.
- Or create new page links is routing.
- By default route routing is inbuilt defined.

- All the routes defined in web.php file of routes

task

- Create two pages and set route for it.

- You can also set roots as following
- Route :: view (Page Name, Route);

Accessing Dynamic Data

- When you want so send data via Route link you need to implement Dynamic data management.

- Example

- Route :: get("/{username}", function($username)

```
 {
        echo $username;
        return view("View Name");
```

- })

- To print data in php page

- Route :: get("/{username}", function($username)
  ```
  {
        echo $username;
        return view("View Name", ["username"=>$username]);
  ```
- })
- * now you can access your data with {{}} in blade file

Task

- Create Routing with anchor tags.

Redirect

- When you want to redirect user request to specific page you just return redirect("page name")

Controller

- Controller is use to connect all the parts of MVC.
- Controller is brain of any Framework.

Create controller

- Go to command prompt
- Php artisan make:controller controllerName
- Controllers are stored in app->http->controllers.
- You must need to create functions in controller.

- Call controller you need to create route in web.php route file.
- You need to import component in web.route file of Laravel
- Import controller with use keyword

  ex. Use App\Http\Controllers\ControllesrName
- Route::get(“URL”, [Controller Name:: class, controller function name]);

- *in older version of Laravel following code is used to run controller
- Route::get(“URL”, ‘ControllerName@functionName’);

Pass parameters to controller with URL.

- You can also pass data in controller via URL parameters

- Example

Route::get(parameterName/{variable Name}, [ControllerName::class, ‘function Name’]);

To print received data you need to use function parameters.

Laravel View

- View of framework is used to store complete HTML code of our application. You can call view from router and controller also,
- You can create Views in resources -> views folder of Laravel

Blade

- Blade is template engine provided by Laravel. Blade in not a part of core MVC Architecture it just use to fast rendering of code.

Call view from Route

- In web.php
- Route::view("URL", "View Name");

Or

- Route::get("URL", function(){

  return view("View Name");

})

Task

- Pass some dynamic data to view with Route.

Call view from controller

- Create a controller
- Php artisan make:controllerName
- Create function in controller

```
Function functionName()
{
	return view("View Name");
}
```

- Import controller in web.php
- Create link to access controller

Task

- Pass Dynamic Data in Controller and print in blade.php file.

Laravel Components

- Components added from Laravel 7 and more.
- Components is most important and advance topics in Laravel.

- Component is make reusable codes like header and footers of website.

How to create a component

- Php artisan make:component component_name

- New Component have two files
  - Php file
  - HTML file

  - Resources views ComponentName.blade.php (for HTML Code of component)
  - App view ComponentName.php (for PHP code of Component)

How to use component in Laravel

- Create New view in Resources -> views - > ViewName.blade.php
- Create Routes for newly created Blade File
- To use Component in View you need to write as following
- <x-Component Name />
- Put some codes in component file

To pass some data in Component

- <x-componentName parameterName=“Value”/>

- To access parameter value receive parameter in constructor of component php file. Store received parameter in variable and use in blade template file with {variable name} syntax.

Task

- Create header component and use in multiple view pages in Laravel.

Laravel Blade Template

Introduction

- Blade template engine is used to convert plain text to php.
- You don't need to write php code every time.
- Just put {{ php code }}
- Provide easy way to create header and footer for reuse.
- Easily add Jquery and Java Script Code

Example

- Create a simple controller and view.
- Create some function and return view from controller.

Expression

- Laravel allows you to put PHP code anywhere in your blade file with just use of {{ }}

- Ex, {{ 10 + 20 }}

- You can receive parameters from controller Routes.
- You can pass arrays from route and Access in blade template files.

Conditional codes

- You can easily add conditional code in template file as following

```
@if(expression)
Executable code
@endif
```

```
@if(expression)
      Executable code for true condition
@elseif(expression)
      Executable code for false condition
@else
      Executable code for false condition
@endif
```

For loop in blade file

```
@for(loop code)
    executable code
@endfor
```

For each loop

- Create some array data and pass from controller.

```
@foreach($tmp as $array)
      executable code
@endforeach
```

Include view in another view

- Create a simple blade file.
- Add some html code

- Use @include("required blade template file name")

Add JS in blade file

- Add <script> tag in your blade file.
- Normal JS code works as expected
- When you need to run some php data you need to convert it.

- Get Array from php controller.
- Put script tag in your blade file

  Let data = @json(php array)

CSRF

- CSRF is used to provide data protection.
- use @csrf
- The csrf tocken is hidden by default.
- **Cross Site Request Forgery**

Comments in Baled Template

- Must use {{-- --}} for comment in blade template file

HTML form

- This Section cover how to get data of form in Laravel.
- We want to male controller.

  create some function in controller to get form data.

  create route for newly created component.
- Then we design form in new blade file.

  put controller name as form handler in action of HTML form and set method to POST.

  *you must specify method POST while create route in web.php file.

- Create path for get view
- You must need to pass token in form as following @csrf.

- Now make request object as parameter in controller of function.
- Now you can print request data as following
- $requestobject->input()

Boot Strap

- 3 ways to add bootstrap in Laravel.

1. You can add CDN. (copy CDN link and paste in head section of your blade file)
2. Or add a common file for CDN links and paste every where when you want to use bootstrap.

Middleware

- When we want to some conditions or restrictions related logics in our application then middleware comes in picture.

Types of Middleware

- Global Middleware
- Grouped Middleware
- Routed Middleware

- Create some views in resources like (users, checkage, noaccess).
- Put some text in every views

Command

- Php artisan make:middleware middleware Name
- Middlewares are stored in Http->middlewares
- Put some logic in middle ware
- You need to register middle ware in kernel file of app->http folder.

Global middleware

- Global middleware is used to apply on all the web pages of your application.
-

Grouped Middleware

- When you want to add middleware on some specific routes only you need to apply routed middleware.
- Php artisan make:middleware middlewareName
- Put some logic for middle ware

- Select middlewareGrouped while register middleware in kernel file
- Add following

```
‘protectPage’=>[
        Select middleware
        \App\Http\Middleware\MIddlewareName::class
]
```

In web.php file

```
Route::group([‘middleware’=>[‘protectPage’]], function(){
	Route::view(‘Route Name’, ‘View Name’);
});
```

Routed Middleware

- Routed Middleware is used to apply to set middle ware on specific routes only.

- To apply middleware create middleware and set function for logic.
- Add some blades files with routes.
- Register newly created middleware in RoutedMidleware section in kernel.php of middleware folder

- Like following
- ‘middlewareName’=> middleware class name

To apply a middleware

- Web.php fille
- Route::view(“URL Name”, “View Name”)->middleware(‘middlewareName’);

URL Generation

- When you work with project you need to check how much data in query string. Or get address last visited page.

Current URL.

- Create some blade files and route
- Add routes in web.php

- Add following to your blade file for get current URL.

- {{URL::current() }}

Full URL

- Create some blade files and route
- Add routes in web.php

- Add following to your blade file for get current URL.

- {{URL::full() }}

Create Link with URL

- <a href='{{URL::to('blade template name')}}'>Link Name</a>

Get Previous URL

- {{URL::previous()}}

Task

- Task 1 : Create form and send data on specific URL
- Task 2 : Create Back button with use ot URL::previous().

Database

Database Connection with mySQL.

- To configure database connection with Laravel you need to configure .env file.
- Select database section and set database, username, password for our database server.

- Now go to config folder and database.php file.

- Create controller for manage database.
- Create some function in newly created controller for getting data from database.
- Add controller in route.

Controller code

```
Use Illuminate\support\facades\DB;
Class ControllerName extends Controller{
    function index(){
        return DB::select("select * from table_name");
    }
}
```

Model in Laravel

- Model is used to connect Laravel project to database.
- We can connect our Laravel application to database by modal or by database classes.

- In previous practical we connect database without use of model and this section we will learn how to connect database to our Laravel application with use of modal.

- In modal we map our database tables to our classes its known as model

- Create class name as defined table name your database, we also define our database structure in our modal and we implement our business logic in our model class

- Users = user
- * users is our table name and user is class name so table name is always plural and modal name is always singular

- Php artisan make:model modelName

- Php artisan make:model user
- * for users table of our database (models are stored in app -> http -> models)

- You need to create controller for our model
- Php artisan make:controller controllerName

- Import model in your controller.
- Create function in your controller.

```
Function getData(){
      return modelName:all();
}
```

- Create route for our controller.
- Route ::get('/URL Path', [Controller Name :: class, 'function name']);
- * you must configure database details in .env file

- You can re allocate table from model just create
- public $table=“target table name”
- Now you are able to get data from new table.

http client

- We can consume and other backend application in our Laravel application via http client

- http client is used to access API data in Laravel. Or Laravel is used to get data from other APIs.

- For testing purpose : `https://dummyjson.com/products`

- Create controller in Laravel
- Create route for our controller.
- Create function in our controller
- use\Illuminate\Support\Facades\Http;

Code of controller functions

```
Function functionName(){
        $data = Http::get(API URL);
        return view(‘view blade name’, [‘collection’=>$data[‘key name’]]);
}
```

*create view to get proper formatted data from API.

Create table in our view and print data received from API.

In view file

```
@foreach  ($collection as $row)
      <tr>
            <td>{{column Name 1}}</td>
            <td>{{column Name 2}}</td>
            <td>{{column Name 3}}</td>
            <td>{{column Name 4}}</td>
            <td>{{column Name N}}</td>
      </tr>
@endforeach
```

HTTP Request Method

- http request is used when you submit, access, download some data from your websites.

- HTTP stands for Hyper Text Transfer Protocol.

Types

GET	:	Used to get Data
POST	:	used to send data
PUT	:	Update Data
DELETE	:	Delete Data

- Create view with one form
- Create controller to handle submitted form data
- Create Route for our form handler controller.
- Add action as formcontroller URL.
- Must set Proper Method in form and Route also.

Controller

- Create object of request in form handler method.
- @csrf add when you send data via post method
- PUT used to update data

Use put

- {{ method_field('PUT')}}

- Form method must be post
- This method adds two hidden fields in our form.

Flash Session

- Flash session is used to store data for single use.
- When we reload our page flash session is destroyed
- Used for display messages for one time.

- Create view and controllers for session
- And create routes for both

- Create form in view blade file with single input field and get some data from user.
- Set method to post and redirect data to controller
- Get input filed data in any variable of function which is declared in our controller.

Code for blade file

```
@if (session(‘session key’))
<h3>Data Saved for {{session(‘session key’)}}</h3>
@endif
```

Controller code

```
public function function_name($request $request){
        $data = $request->input(‘filed name’);
        session()->flash(‘session name’, $data);
        return redirect(‘view blade name’);
}
```

Task

- Create login form and get data from user
- Check if the username and password is admin then set success flash session and if not match then set error flash session
- Must redirect user to proper page.
  - Home for success
  - Login for error

Login Session

- Session is normally used to store user authentication and session is accessible in all the pages of our project.

Crate blade file for login form.

- Create login form with email and password.

Controller

- Create controller for handle session code and create routes for html form and controller file.

Task

- Create login form and get data from user
- Check if the username and password is admin then set success session and if not match then set error session.
- Must redirect user to proper page.
  - Home for success
  - Login for error
- Create about, contact, projects page, mentioned page is accessible if the session is set if session is not valid then redirect to login page.
- Also create logout to remove session from system.
- Also create grouped middleware to check sessions on specifics routes only.

```
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
class logincontroller extends Controller{
    public function loginProcess(Request $request){
        $uname = $request->get("uname");
        $upass = $request->get("upass");

        if($uname == "admin" && $upass == "admin"){
            session()->put('username', $uname);
            return redirect("home");
        }else{
            session()->flash('msg', 'Invalid Username or Password');
            return redirect("login");
        }
    }
}
```

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

Route::view("/login", "login");

Route::post("/loginprocess", [logincontroller::class, 'loginProcess']);

Route::group(["middleware"=>["sessionCheck"]], function(){
    Route::view("/profile", "profile");
    Route::view("/home", "home");
    Route::view("/data", "data");
});
```

```
Route::get("/logout", function(){
    if(session()->has('username')){
        session()->pull('username');
    }
    session()->flash('msg', 'Logout Successfully');
    return redirect("/login");
});
```

File Upload

- File upload is very common task for all the web developers.
- Create view with form for select file to upload.
- Create controller for uploading process.

```
function uploadProcess(Request $request){
	return $request->file('file controller name')->store("Folder Name");
}
```

Uploaded files are found in

Project -> storage -> app ->

* storeAs function is used to store file with specific name.

Localization

- Localization is make our website for more than one human languages.

- Create some blade file
- Add routes for all

- Go to resources -> lang folder
- Create php file in desired language folder
- Make folders for other languages and put same file in all fodders.

- Put every language file code like following.

```
<?php
      return [
            “Key1”=> Data,
            “Key2”=> Data
      ]
?>
```

- To get languages in blade file
- {{ __('page Name.array Index')}}
- * you can set default language from config.php set locale = default language

Web.php

```
Route::get('URL/{lang}', function($lang){
        App::setLocale($lang);
        return view('page name');
})
```

Get Data From Database

- Create database table named users and add some user data in table.
- Create blade file to display userdata
- Create controller to get data function
- Create model to manage database
- Get all data in controller via modal
- Pass all data in view and print with foreach loop.

Get data with Pagination

- We can get data from database and display result with pagination format.
- Do get data code as per previous code.

- Add following code while fetch data in controller

```
$data = ModelName::paginate(count number of records);
```

Next and previous

```
<div>
      {{$resultset->links()}}
</div>
```

* ,w-5 class shows arrow in pagination you can edit as per your rrequirment

Task

- Apply Bootstrap style on pagination links.

Add Data in Table From HTML Form.

- Create view with Required HTML elements.
- Create controller for handle form request.
- Create required routes.
- Create Model for our Database Table.
- In Controller import table model and create instance of our newly created Model.

Controller function

```
Public function addData(Request $request){
        $modelObject = new ModelObject;
        $ modelObject->fieldname1 = $request->formFieldName1;
        $ modelObject->fieldname2 = $request->formFieldName2;
        $ modelObject->fieldname3 = $request->formFieldName3;
        $ modelObject->fieldname4 = $request->formFieldName4;
        $ modelObject->fieldname5 = $request->formFieldName5;
        $ modelObject->save();
}
```

- By default Laravel required created_at, updated_at named two special fields in every table of database.

- To resolve this add in model controller file
- Public $timestamps = false;

Delete Data from Table.

- Create link to delete a data with list created in previous example.
- <a href=“controllername/data”>Delete</a>
- Make controller for delete data controller

Code for controller

```
Public function deleteData($id){
    $data = $modelname->find($id);
    $data->delete();
    // code for return to view
}
```

Update Data

- Update existing data in database from Laravel.
- Add new column for edit link in your exiting data list.
- Create route for data show
- Create function in controller for get data from specific record,

```
function findData($id){
        return modelName :: find($id);
}
```

Create view for update form

- create blade file with form to show pre filled data of selected user.
- Print all values in form
- Create route for update user data.

```
function updateData(Request $request){
    $data = ModelName::find($request->id);
    $data->column1  = $request->form field 1;
    $data->column2  = $request->form field 2;
    $data->save();
    Return to view;
}
```

Query Builder

- Query builder is used to access data from database without use of models
- We have two types of connectivity in Laravel for access data from database
  - With use of Model
  - With use of DB class (query builed)
- Set database configuration in .env file.
- Create some controller Create function to get data
- Also Add Route for controller.

- Use use\Illuminate\support\facedes\DB in your controller
- To get data from tabe
- Return DB::table(“Table Name ”)->get();

where

- Return DB::table("Table Name ")
- ->where(column name, value)
- ->get();

Find

- Find for specific ID
- Return DB::table("Table Name ")->find(id);
- To convert data in array
- Return (array)DB::table("Table Name ")->find(id);

count

- Return DB::table(“Table Name ”)->count();

Insert data

```
Return DB::table(“Table Name ”)->insert([
        ‘key1’ => value1,
        ‘key2’ => value2
        ‘key3’ => value3
        ‘keyN’ => valueN
]);
```

Update Data

```
Return DB::table(“Table Name ”)
->where(‘column name ’, value)
->update([
        ‘key1’ => value1,
        ‘key2’ => value2
        ‘key3’ => value3
        ‘keyN’ => valueN
]);
```

Delete

Return DB::table("Table Name ")->where('column name ', value)->delete();

Aggregation in DB class

- Aggregation is must required for access database for find max, min, avg, sum and etc.

- Create some controller and make function for execute query.

Sum function

- Return DB :: table(table name)->sum(column name);

Max Function

- Return DB :: table(table name)->max(column name);

Min Function

- Return DB :: table(table name)->min(column name);

Avg Function

- Return DB :: table(table name)->avg(column name);

SQL Joins

- Joins is generally used to get data from more than one table based on some same columns in every tables.

- Create controller and make Route for it.
- Import DB in controller
- Create function for get Data.

Get Simple Data

Return DB :: table(table name)->get();

Simple Join

Return DB :: table(table name)

->join(second table name, first table name. column name, “operator”, second table name. column name)

->get();

Select Specific Column

Return DB :: table(table name)

->join(second table name, first table name. column name, “operator”, second table name. column name)

->select(table name.column name, table name.column name)

->get();

With Where

Return DB :: table(table name)

->join(second table name, first table name. column name, "operator", second table name. column name)

->where(table name. column name, value to be compare)

->get();

Migration

- Migration is Most important topic in interview.

- Migration is used to alter table programmatically.
- Create file for database table and just migrate it will make a merged table.

To create table with migration

- Php artisan make:migration create_tablename
- Above command create an entry for new database table and you will entry at following path
- Database -> migration -> table file.

- You find two function in migration file up() and down()

- Up() fun for create table
- Down() run for delete table.

- Every table have two default columns id and timestamp
- You can also define new columns like following.

Sample Code

```
Public function up()
{
        Schema::create(table name, function(Blueprint $table){
                $table->id();
                $table->string(‘fname’);
                $table->string(‘lname’);
                $table->string(‘city’);
                $table->timestamp();
        })
}
```

*must set database configuration.

Commands

- Php artisan migrate
- * Laravel manage migration status in migration table.

Advance Migration

- Here we learn some advance migration commands like reset, rollback, refresh and single file migrations.

- *must configure database in .env file

Basic Command

- Php artisan migrate
- Migrate all the tables with default table of Laravel

Reset Migration

- Reset migration is used to remove all migrations from Laravel.
- Php artisan migrate:reset

Rollback Migration

- Php artisan migrate rollback
- Php artisan migrate:rollback --step 2

Refresh Migration

- Remove all migrated table and recreate it.
- Php artisan migrate:refresh

Single table migration

- Php artisan migrate --path ./ dtabase / migration/ file path

Jet Stream

- Jet Stream is used to create complete authentication system with sign up and login forms.
- * must check version of Laravel we need Laravel version 4+ we then we easily use Jetstream in Laravel.
- Laravel –v for check version of Laravel.

To update Laravel to latest version

- Composer global remove Laravel/installer
- Composer global require Laravel/installer

New project

- Laravel new project_name –jet
- You have asked about two options when you use Laravel jet
  - Livewire
  - Intetia
  - Choose livewire

Wait for complete installation.

- Run following commands

Npm install

Npm run dev

- * must set database in .env file.

- Php artisan migrate
- After successfully migration you will find login and register links on home page.
- * must set app Url to http://localhost:8000

Accessors

- Accessors is used to modify data before pass database data to your view file.

- Create controller with function.
- Create model for our database table.
- Create route for our controller.

Model file

- Create a function for accessor.

```
Function getFiledNameAttribute(value){
      return ucFirst($value); // uppercase first letter
}

Function getFiledNameAtteribute($value){
      return desired operation.
}
```

Mutators

- Mutators is used modify data when you insert data in database table.
- Create controller and model for access database.
- Create route for our controller
- Manually Send some data in table.

```
Class modelController extends Controller{
        function functionName()
        {
                $modelObject = new Model;
                $modelObject->filed name = “Value”;
                $modelObject->filed name = “Value”;
                $modelObject->filed name = “Value”;
                $modelObject->save();;
        }
}
```

```
Function setFiledNameAttribute($value){
        return $this->attributes[filed name] = value with modification.
}

Public $timestamps = false;
```

One to One Relationship

- When you want to get data from more than one table we need to implement relation.

- One to one relationship is used to map table 1 id to table 2 id.

Types of relationship

- When table id mapped with another single table its known as one to one relationship.
- When table id mapped with multiple table its known as one to many relationship.
- When multiple table id mapped with single table its known as many to one relationship.
- When multiple table id mapped with multiple tables its known as may to many relationship.

- Create two tables and create models for both tables
- Create controller with function for getData
- Create route for controller.

- Create function like following in primary table model

```
Function getSomeData(){
      return $this->hasOne(‘second table  model name’ :: class);
}
```

In controller

- Return Model::find(id)->second table function. (without () sign)

One to Many

- Single record of table 1 is mapped with multiple records of second table.

- Create controller with function for get data
- Create route of our controller
- Create models for required tables.

For parent model

```
Function function_name(){
      return $this->hasMany(‘secondary model path’);
}
```

Fluent strings

- Fluent string is used to apply string function very quickly
- Specially we can apply many string function in chaining.

- You need to import string as following.

Use Illuminate\Support\str;

```php
$data = "welcome to world of fluent string in laravel";
$data = Str::ucfirst($data);
$data = Str::upper($data);
$data = Str::lower($data);
$data = Str::camel($data);
$data = Str::replaceFirst("welcome", "Shyam", $data);
$data = Str::of($data)->ucfirst($data)->replace("to", "Hello");
$data = Str::before($data, "to");
$data = Str::after($data, "to");
$data = Str::repeat($data, 2);
$data = Str::words($data, 5);
$data = Str::between($data, "to", "in");
$data = Str::contains($data, "to");
$data = Str::startsWith($data, "welcome");
$data = Str::endsWith($data, "laravel");
```

Route Model Binding

- Route model binding used to get data from database with use of route and model only,

- Create model for our table
- use model in web.php
- Create function in web.php route

```
Route::get("/url", [controller::class, 'function name']);
```

Now time to modify our url to receive id with request and get data from table

```
Route::get("/url/{id}", [controller::class, 'function name']);
```

Or

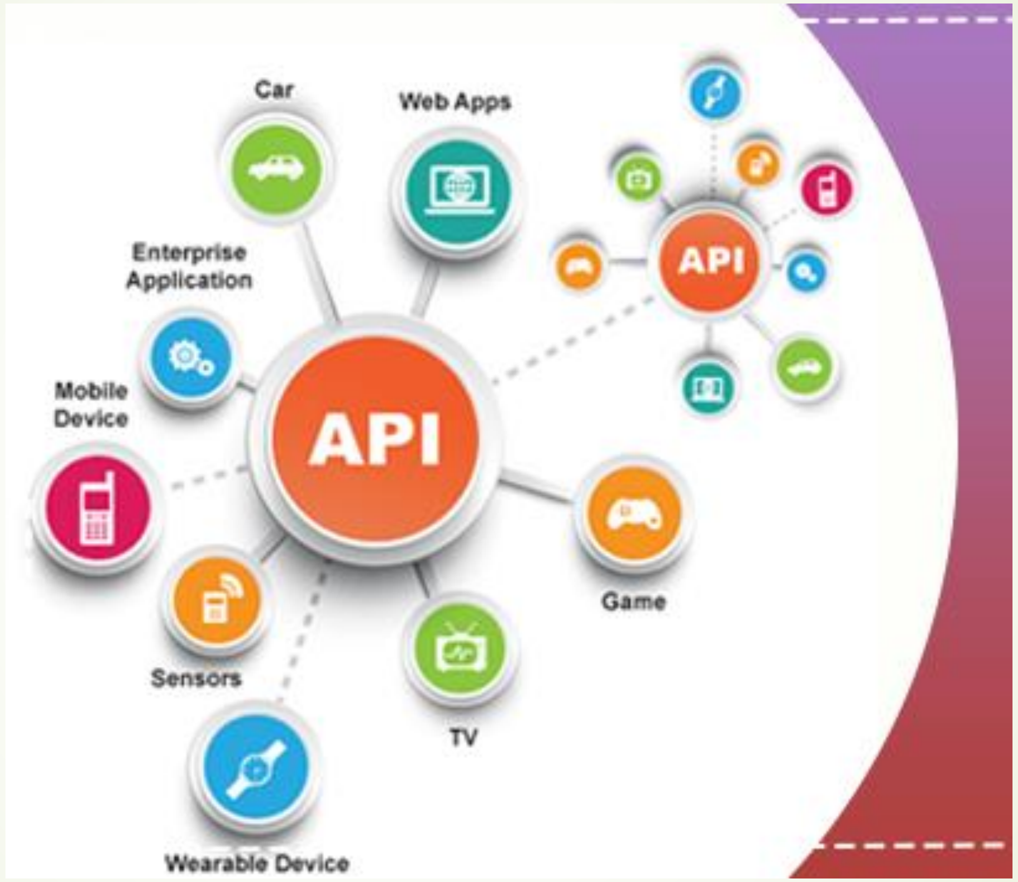
```
Route::get("/url/{id:name}", [controller::class, 'function name']);
```

Web.php code

```
// with controller function
//Route::get("/getdata/{id}", [studentController::class, 'index']);

// call with Model
Route::get("/getdata/{id}", function(student $id){
    return $id;
});
```

API


Car
Web Apps
Enterprise Application
Mobile Device
API
API
Game
Sensors
TV
Wearable Device

- API is used to send or receive data from two or more same or different technology.
- Like PHP – Android
- An application programming interface (API) is a way for two or more computer programs to communicate with each other.
- API is Mainly used to get data in frameworks like Angular, react and many more witch is not able to connect database directly.

- API is used to receive and send data in JSON format to any technology JSON is plain text format for data store and exchange.

- JSON is an open standard file format and data interchange format that uses human-readable text to store and transmit data objects consisting of attribute–value pairs and arrays. It is a common data format with diverse uses in electronic data interchange, including that of web applications with servers.

- Use postman for testing API

First API in Laravel

- Create controller with function.
- Create route for our controller. * must create API Route in

  Routes -> api.php file.

  import controller in api.php file.
- Example.

  Route::get('URL', [controller class :: class, 'function name']);

*get API is used to get data from API.

- You must need to add api in every url while using some api as following
- http://localhost:5000/api/apiname
- All the data are transferred in JSON format.

Controller Function

```
Function functionName(){
	return [“name”:”Harmeet”, “lname”:”Purohit”];
}
```

Send data in array format laravrel convert array to JSON.

Get API in Laravel

- Create Controller with function to access data.
- Create Model for access our data of database table.
- Configure .env file for database
- Create route in api.php
- Import controller in api.php
- Import model in our controller.
- Return all data from model with our controller function

API with Parameters

- When you want to get data from some specific id we need to implement parameter with API.

- Add parameter in route of api.php
- Also receive data in controller function.
- Also write logic for run API without parameter.

* you need to add ? After parameter name to make parameter optional.

Controller function

```
Public function getData($id = null)
{
    return $id ? Model::find($id) : Model::all();
}
```

POST API.

- Post API is used to save data to Database via API.

- Create controller with function to save data
- Create model for our table.
- Create route in api.php file
- Extract all data from request in controller class
- Create object of model and save data on all fields
- Return success message to api

Search API

- Search API is used to find any specific data from database

```
Function searchData($name)
{
    return modelName :: where (Column name, $name)->get();
}
```

Like query

```
Function searchData($name)
{
    return modelName :: where (Column name, “like”,”%”.$name.”%”)->get();
}
```

API Validation

- Validation used to validate data received in API.
- Create form for add data
- Create controller with function
- Create model for database table
- Create route for our API.

Create API URL for validation

```
Route::post("/urlname", [controller class name :: class, 'function name']);
```

Controller function

```
Use Illuminate\Suport\facede\Validator;
Public function function_name(Request $request){
    $rules = array(
        “filedname” => “Rule”;
        “filedname” => “Rule”;
        “filedname” => “Rule”;
    );
    $validator = Validator::make($request->all(), $rules);
If($validator->fails()){
        return $validator->errors();
        or
        return response()->json($validator->errors(), error code);
}
else
{
        another action
}
}
```

Rules

- Required
- Max : length
- Unique

API with Resource

- Resources used with CRUD operation
- Laravel automatically create function for crud and managed by only single route,

- Create controller with --resource switch
- Create model
- Configures database

- --resource switch create required function in your controller.

Route for Resource in api.php

Route::apiResource("/url", controller name);

All the built-in function called based on request method.

Pre generated function with resource controller is Index(), Create(). Store(), Show(), Edit(), Update (), Destroy()

API Authentication

- Laravel API authentication with use of Sanctum.
- API Authentication is used to secure our API for unauthorized access of our API.
- Must configure database in .env

- Install Laravel Sanctum

Composer require Laravel/sanctum

- **Publish**

```
php artisan vendor:publish --
provider="Laravel\Sanctum\SanctumServiceProvider"
```

- **Migrate**

Php artisan migrate

- **Add the Sanctum's middleware.**

Use Laravel\Sanctum\Http\Middleware\EnsureFrontendRequestsAreStateful;

```
protected $middlewareGroups =
 [
      'api' => [
            EnsureFrontendRequestsAreStateful::class,];
]
```

- **To use tokens for users.**

```
use Laravel\Sanctum\HasApiTokens;
class User extends Authenticatable{
	use HasApiTokens, Notifiable;
}
```

- **Let's create the seeder for the User model**

Php artisan make:seeder UserTableSeeder

```
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Hash;
DB::table('users')->insert(
[     'name' => 'John Doe',
       'email' => 'john@doe.com',
      'password' => Hash::make('password’)]
);

php artisan db:seed --class=UsersTableSeeder
```

- Create controller

```
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use App\Models\User;
use Illuminate\Support\Facades\Hash;
class UserController extends Controller
{
    function index(Request $request)
     {
        $user= User::where('email', $request->email)->first();
        if (!$user || !Hash::check($request->password, $user->password))
        {           return response([
                   'message' => ['These credentials do not match our records.']],
                   404);
        }
$token = $user->createToken('my-app-token')->plainTextToken;
            $response = [
                'user' => $user,
                'token' => $token
          ];
          return response($response, 201);
     }
}
```

URL

- Create URL in API.php file
- Import user controller in api.php file.
- Route::post(“login”, [UserController::class, ‘index’]);

For Testing URL

- Open postman
- Set method to POST
- Localhost:8000/user
- Send data

```
{
    “email”=>”Your Registered URL”,
    “password”=>”Your Password”
}
```

- **Test with postman, Result will be below**

```
{
    "user": {
        "id": 1,
        "name": "John Doe",
        "email": "john@doe.com",
        "email_verified_at": null,
        "created_at": null,
        "updated_at": null    },
         "token": "AbQzDgXa..."
}
```

- **Make Details API or any other with secure route**

```
Route::group(
            ['middleware' => 'auth:sanctum'], function(){
            //All secure URL's
      });

Route::post("login",[UserController::class,'index’]);
```

* Go to api.php and set auth method to sanctum

Try to access with token

- Create new route for get data
- Try to call any API from server

- * you will get error 500 internal server error.

- You need to pass your authorization token with your request as following.
- Go to header of postman API call Section and add key authorization and value Bearer Your Token

Secure All API

- Add your all API Routes between

```
Route::group(['middleware' => 'auth:sanctum'],
    function(){
        //All secure URL's
    });

Route::post("login",[UserController::class,'index']);
```

Upload Files with API

- Create controller for upload files with function with Request object as Parameter.
- Add route for API with POST method.

```
Function uploadProcess(Request $request){
       return $request->file(‘file controller name’)->store(‘path’);
}
```

Choose file from postman and perform file upload.

(go to body of postman request, select key enter key name, change type to file, in value select file to upload)

- return $request->file('file1')->store('images');
- return $request->file('file1')->storeAs('images', 'abc.jpg');
- return $request->file('file1')->storeAs('images', $request->file("file1")->getClientOriginalName());

Custom Commands

- Laravel provide to create custom commands for run custom codes

- Php artisan

To view all commands.

- Php artisan make:command command_name
- App -> console -> commands -> command file name
- Set command name
- Handle function used to handle logic of command.

- Register command in kernel.php comamds folder.

```
Protected $commands = [
      Commands\Command file Name :: class
]
```

Example

```
Public function  handle(){
	$this->info(DB::connection()->getDatabaseName());
}
```

Connect Multiple Database

- Configure database in .env file
- Create controller for handle database functions
- Create model for required table.

- Replicate database configuration in .env file for connecting second database and set some different property name for new database connection.
- Open config folder and go to database.php replicate connection for mysql and set database property to new property.
- Create model and return table data as following.

```
Public function index(){
        return DB::connection(connection name)->table('table name from any
database')->get();
}
```

With model

- Create model for access database table data.
- Import models in controller.

Return modelName::all();

Go to model

- And set
- Public $connection = ‘new connection name’

Upload project with C-Panle

- Upload Laravel project with c- panel
- Upload .zip file in public_html of our project.
- Extract uploaded file in public_html
- Run the project and view errors.
- Moves index and other files form public of Laravel to root of public_html
- Go to index.php

- For auto loader, run the application Set the path to parent folder

Laravel new Feature in Laravel - 9

- You need PHP 8 for Laravel 9
- Laravel new release every year.
- Laravel 9 is backward comparable

- Symphony mailer (previous versions use shift mailer for mail send)
- Flysystem 3.x used for file system.
- Improve accessor and mutators (combine both in same functions)
- Controllers Routed Group
- Full text index for database
- WhereFullText clause in database
- Scout Database engine for very small application and light weight
- Rendering inline blade file
- Slot name shortcut

- Bootstrap pagination
- Improve error page description
- Detailed routed list
- Routed bind with enum
- Automation testing for project
- New helpers str and to_route

Slots in Laravel

- Slots are introduced from Laravel 8.
- Slots are used to send data from view blade file to any component file
- Component is reusable code in Laravel like header, footer of website and etc.

- Unnamed data
- Named data

- Write some content in between of component and access in blade file with {{$slot}} keyword.

- Give name to every slot and set some name loke following.
- <x-slot:name>Content<x-slot>
- Print with
- {{$slotname}}

Controller Route Group

- In older version of Laravel you need to create routes for every methods defined in our controller class.
- In new features with Laravel you Just create class group route for controller class and access all the route from on declaration.

- Create controller with some function and create API for it.
- Create route in API.php

API.php file

```
Route::Controller(Controller class Name :: class)->group(function(){
    Route::Protocol(“Request Name”, “Controller Function Name”);
    Route::Protocol(“Request Name”, “Controller Function Name”);
    Route::Protocol(“Request Name”, “Controller Function Name”);
    Route::Protocol(“Request Name”, “Controller Function Name”);
    Route::Protocol(“Request Name”, “Controller Function Name”);
});
```

Remove CSRF

- Go to Verify CSRF Token file.
- Add URL to bypass CSRF security from routes.

Inline Blade Templates

- With inline blade templates introduced from Larvael 9.
- Blade is template engine of Laravel.
- Used to convert Dynamic data to HTML.
- Blade template is used generate fast reponse.

- In Previous version of Laravel we need to create view file for every blade.file in view folder of resources.

- In new version of Laravel we create inline blade in controller file.
- Import Blade in your controller file.
- Just return blade from function.

Controller file

```
Public function index(){
    return Blade::render('HTML code');
}
```

* You must need to use single quote while using inline blade template in Laravel.

With Dynamic Data

```
Public function index(){
        $age = 15;
        return Blade::render('answer is {{$age}}', ["age"=>$age]);
}
```

Markdown Mail Template

- Markdown mail template is used to send emails from Laravel with default email template.

- Php artisan make:mail mailtemplatename –markdown=email.samplemail

- Generated files

1 App\mail

2 view\resource\email