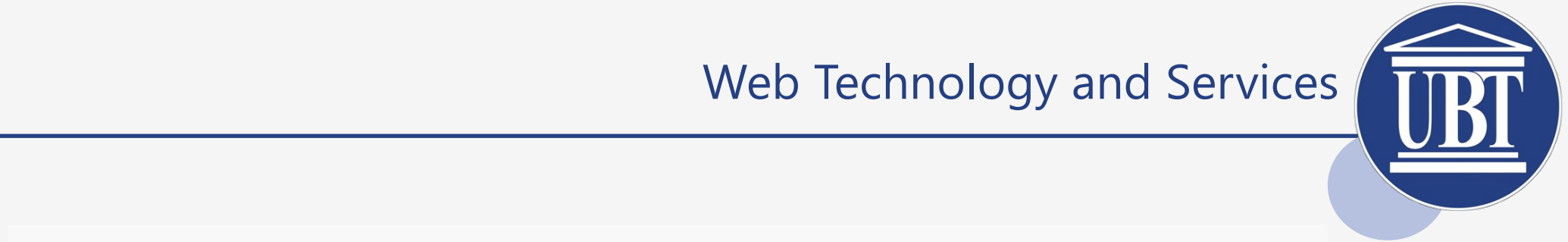


Source: https://www.w3.org/wiki/WebServices

Learning Plan

- Overview
- Installation
- Application Structure
- What is Artisan
- Routing
- Middleware and how to use it ?
- Controllers
- What is blade ?
- Database and Eloquent ORM
- Crud with validation & DB
- Best Practices

Overview

- Powerful MVC Framework
- Created by Taylor Otwell in 2011
- Powerful features
- Incorporates many of the best features of frameworks like CodeIgniter, Yii, ASP.NET MVC, Ruby on Rails, Sinatra, and others.
- Saving time
- Secure Environment

Overview

The web application becomes more scalable, owing to the Laravel framework.

Considerable time is saved in designing the web application, since Laravel reuses the components from other framework in developing web application.

It includes namespaces and interfaces, thus helps to organize and manage resources.

Version	PHP (*)	Release	Bug Fixes Until	Security Fixes Until
6 (LTS)	7.2-8.0	Sep 3rd, 2019	Jan 25th 2022	Sep 6th, 2022
7	7.2-8.0	Mar 3rd, 2020	Oct 6th, 2020	Mar 3rd, 2021
8	7.2-8.1	Sep 8th, 2020	Jul 26th, 2022	Jan 24th, 2023
9 (LTS)	8.0-8.1	Feb 8th, 2022	Feb 8th, 2024	Feb 8th, 2025
10	8.0-8.1	Feb 7th, 2023	Aug 7th, 2024	Feb 7th, 2025

Features

- Modularity
- Testability
- Routing
- Configuration management
- Query builder and ORM (Object Relational Mapper)
- Schema builder, migrations, and seeding
- Pagination
- Migrations
- Security
- Template engine
- E-mailing
- Authentication
- Redis
- Queues
- Event and command bus

Installation

Installation with Docker and Composer

- Laravel uses Composer to manage its dependencies
- Composer is dependency management tool for PHP, like a library full of
- books
- **NOT** like Yum or apt
- Per project tool (vendor folder), not per system

Step 1: https://getcomposer.org/download/

Step 2: php artisan –version

Step 3:
composer create-project laravel/laravel example-app

cd example-app

php artisan serve

Structure

The Root Directory

- The app Directory
- The bootstrap Directory
- The config Directory
- The database Directory
- The lang Directory
- The public Directory
- The resources Directory
- The routes Directory
- The storage Directory
- The tests Directory
- The vendor Directory

The App Directory

- The Broadcasting Directory
- The Console Directory
- The Events Directory
- The Exceptions Directory
- The Http Directory
- The Jobs Directory
- The Listeners Directory
- The Mail Directory
- The Models Directory
- The Notifications Directory
- The Policies Directory
- The Providers Directory
- The Rules Directory

Structure

app: This directory contains the core code of the application.
bootstrap: This directory contains the application bootstrapping script.
config: This directory contains configuration files of application.
database: This folder contains your database migration and seeds.
public: This is the application's document root. It starts the Laravel application. It also contains the assets of the application like JavaScript, CSS, Images, etc.

resources: This directory contains raw assets such as the LESS & Sass files, localization and language files, and Templates that are rendered as HTML.
storage: This directory contains App storage, like file uploads etc. Framework storage (cache), and application-generated logs.
test: This directory contains various test cases.
vendor: This directory contains composer dependencies.

Artisan

Artisan is command-line interface for **Laravel** Commands that are saving time
Generating files with artisan is **recommended**
Run **php artisan list** in the console

#Laravel Sail

If you are using Laravel Sail as your local development environment, remember to use the sail command line to invoke Artisan commands. Sail will execute your Artisan commands within your application's Docker containers:

./sail artisan list

Routing

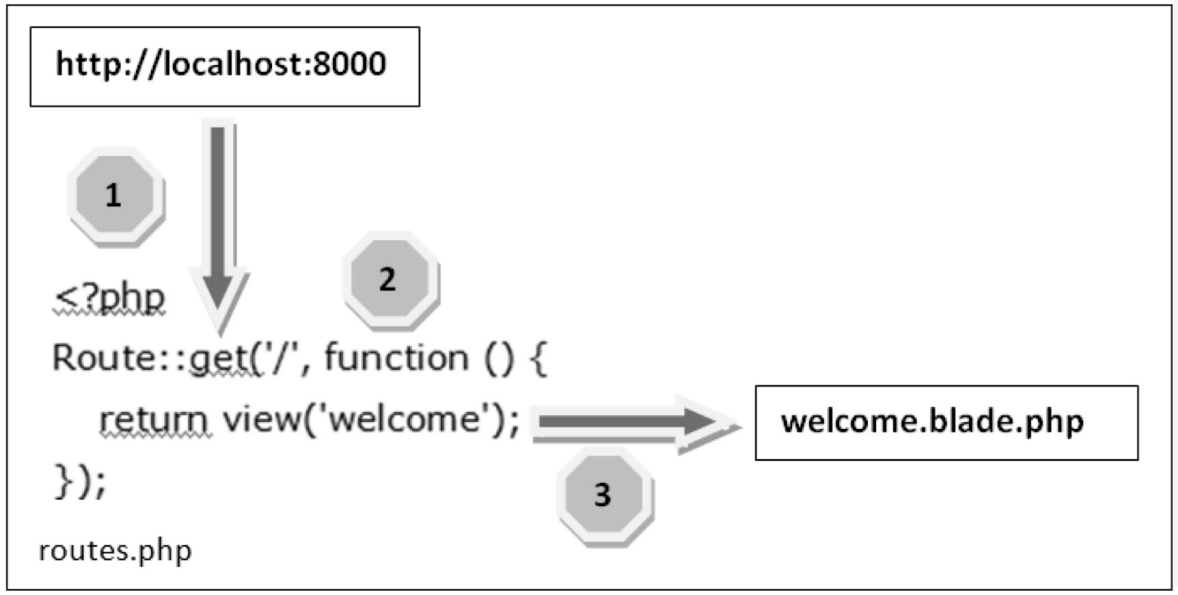
Basic routing is meant to route your request to an appropriate controller. The routes of the application can be defined in **app/Http/routes.php** file. Here is the general route syntax for each of the possible request

```
Route::get('/', function () {
    return 'Hello World';
});

Route::post('foo/bar', function () {
    return 'Hello World';
});

Route::put('foo/bar', function () {
    //
});

Route::delete('foo/bar', function () {
    //
});
```

app/Http/routes.php

```
<?php
Route::get('/', function () {
    return view('welcome');
```

resources/view/welcome.blade.php



Required Parameters

→ Route::get('ID/{id}',function($id) { echo 'ID: '.$id; });

Optional Parameters

Sometimes developers can produce parameters as optional and it is possible with the inclusion of **?** after the parameter name in URL.

→ Route::get('user/{name?}', function ($name = '**FilanFisteku**') { return $name;});

Named Routes

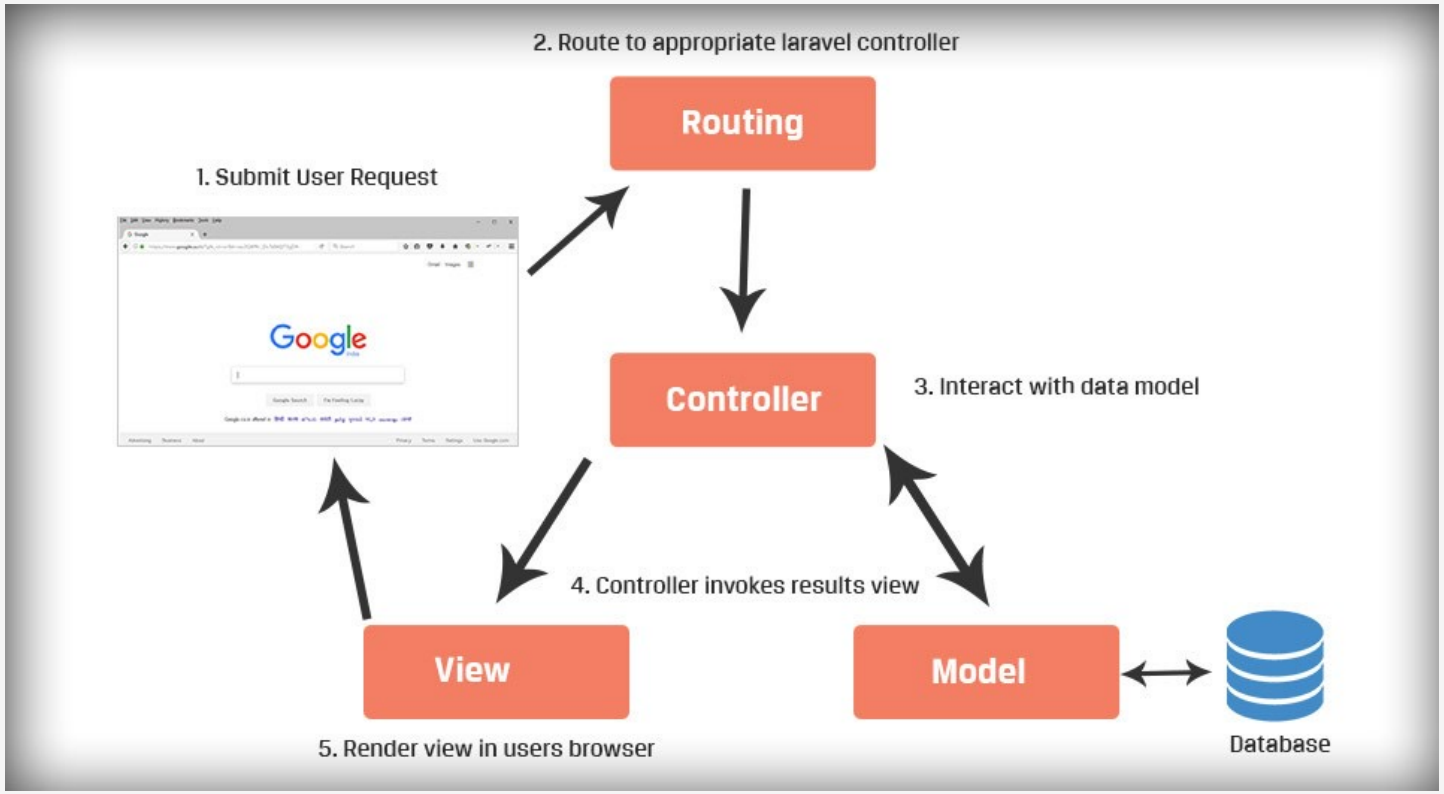
Named routes allow a convenient way of creating routes.

→ Route::get('user/profile', 'UserController@showProfile')->name('profile');

The user controller will call for the function **showProfile** with parameter as **profile**. The parameters use **name** method onto the route definition.

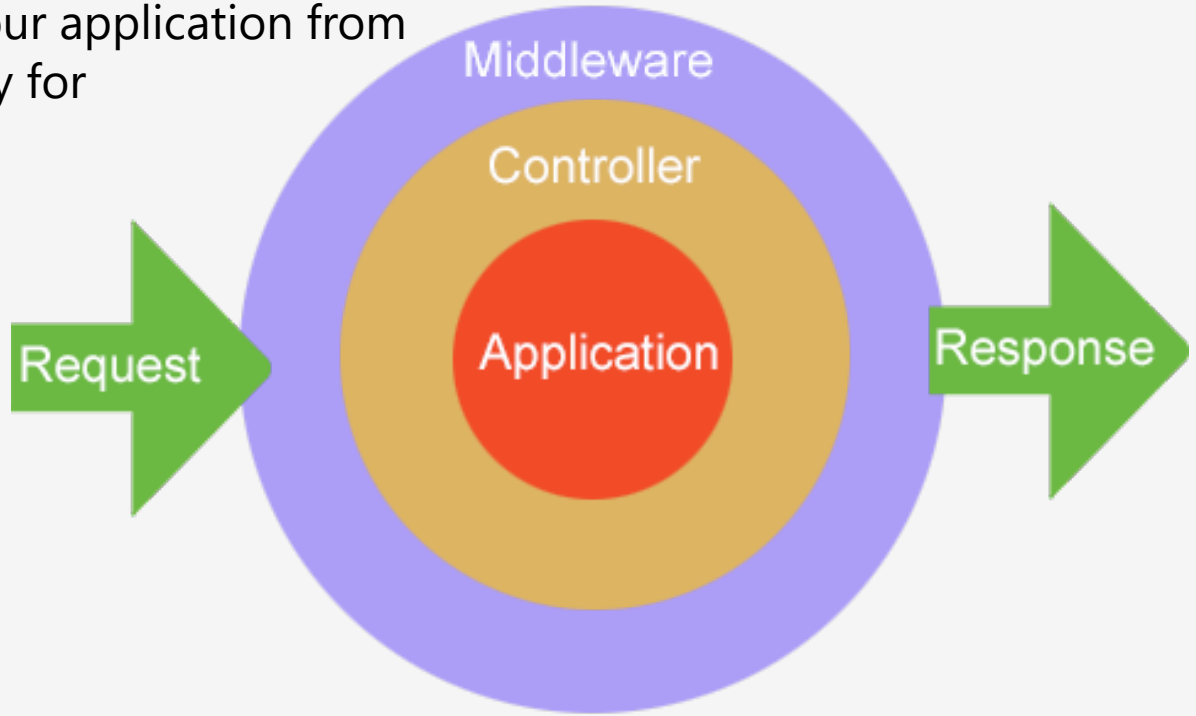
Routing

The best and easy routing system I've seen
Routing per middleware / prefix or namespace
Routing per request method (GET, POST, DELETE, etc.)
ALWAYS name your route !



Middleware

- Middleware acts as a bridge between a request and a response. It is a type of HTTP filtering mechanism
- Laravel includes several middleware's
  – Authentication, CSRF Protection
- The auth middleware checks if the user visiting the page is authenticated through session cookie
- The CSRF token protection middleware protects your application from cross-site request forgery attacks by adding token key for each generated form



Controllers

Instead of defining all of your request handling logic as closures in your route files, you may wish to organize this behavior using "controller" classes.

Controllers can group related request handling logic into a single class.

For example, a UserController class might handle all incoming requests related to users, including showing, creating, updating, and deleting users.

By default, controllers are stored in the app/Http/Controllers directory.

php artisan make:controller UserController --plain

Assigning Middleware to Route

```
Route::get('profile', [ 'middleware' => 'auth', 'uses' => 'UserController@showProfile' ]);
```

Blade Templates

- Blade is the powerful template engine provided by Laravel
- All the code inside blade file is compiled to static html file
- Supports plain PHP
- Saves time
- Better components mobility, extend and include partials
- The blade views thus designed, are compiled

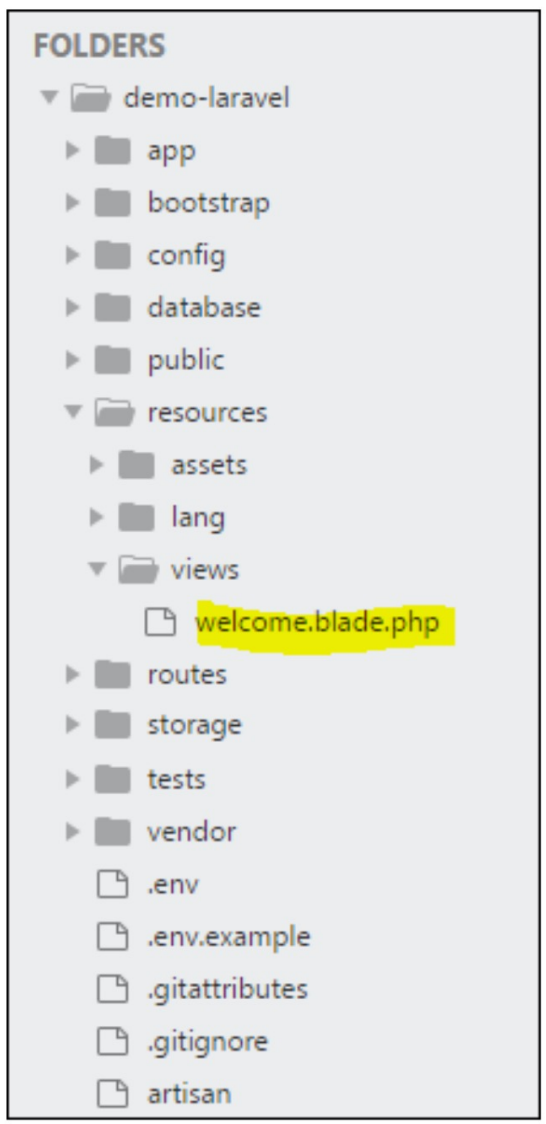
and cached until they are modified.

Master layout

```
<!doctype html>
…
<body>
   @yield('content')
</body>
</html>
```

Child layout

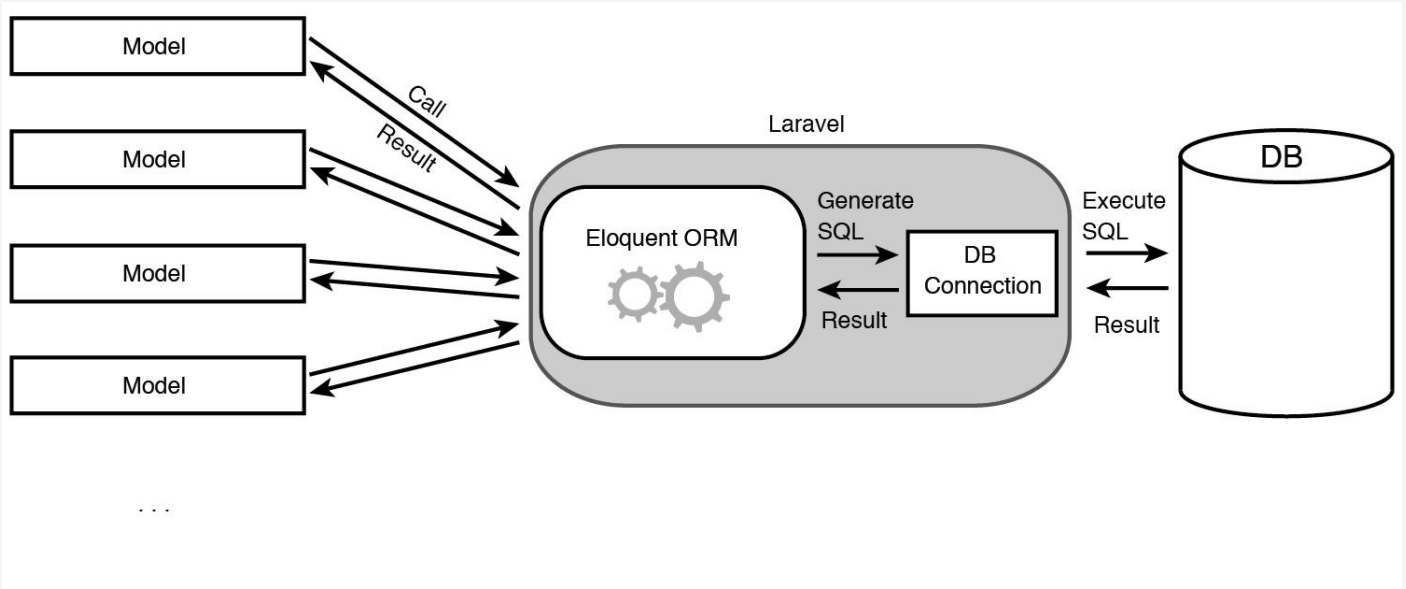
```
@extends('layout.master')
@section('content')
…
@stop
```



Database & Eloquent ORM

The Eloquent ORM (Object-Relational Mapping) Provides Simple Active Record Implementation For Working With The Database

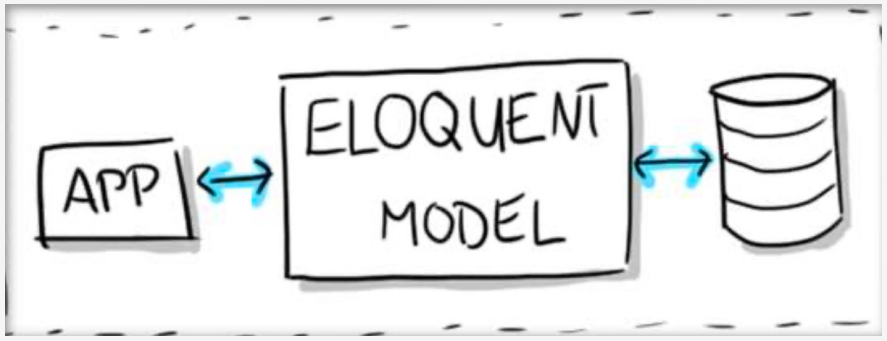
```
$article = new Article();
$article->title = 'Article title';
$article->description = 'Description';
$article->save();
```



INSERT INTO `**article**` (`title`, `description`) VALUES ('Article title', 'Description');

Database & Eloquent ORM

Each Table Has Its Own “Model”. You Can Use The Model To Read, Insert, Update Or Delete Row From The Specific Table



Laravel model

ClassName	Singular of table name	`protected $table = 'custom_name'`
Primary key	id	`protected $primaryKey`
Timestamp	created_at, updated_at	`protected $timestamp = false`
Guarded	array of fields name	`protected $guarded = array('id', 'password')`
Fillable	array of fields name	`protected $fillable = array('id', 'password')`

Best Practices

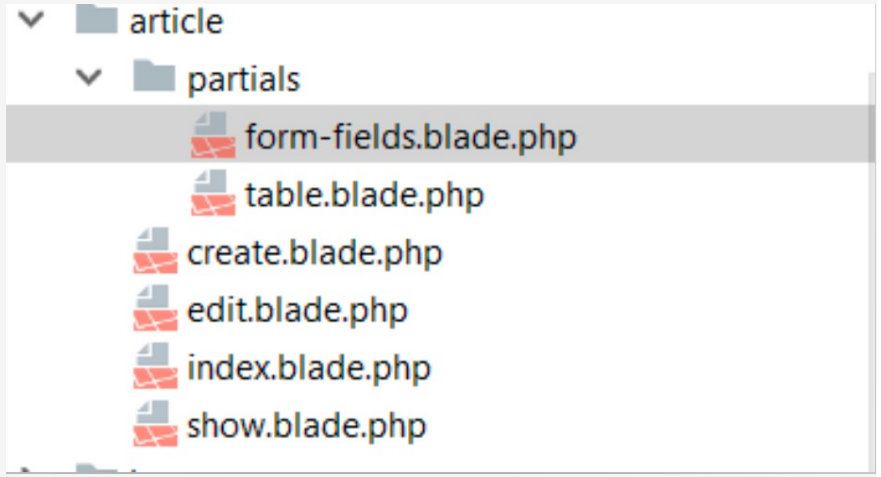
NEVER write queries or model logic inside the controller! The controller job is to communicate with the model and pass data to the view.

```
/**
 * Display a listing of the resource.
 *
 * @return \Illuminate\Http\Response
 */
public function index()
{
    $goodArticles = Article::latestPaginatedArticles( limit: 6);
    $badArticles = Article::orderBy( column: 'created_at',  direction: 'DESC')->paginate( perPage: 6);
```

Best Practices

Extend and include partials. For example share the same form fields on 2 pages – add and edit

```
{!! Form::open(['route' => 'article.store', 'class' => 'form-horizontal']) !!}
    @include('article.partials.form-fields')

    <div class="form-group">
        <div class="col-sm-2 col-sm-offset-10">
            <button class="btn btn-success btn-block" type="submit">
                Add
            </button>
        </div>
    </div>
{!! Form::close() !!}
```



Best Practices

Always use the CSRF token protection that Laravel provides in forms you create, the hackers will not be able to spam your forms and database

```
<form method="POST">
    {{ csrf_field() }}

    <button type="submit" class="btn btn-success">Submit</button>
</form>
```

```
<form method="POST">
    <input type="hidden" name="_token" value=
    "XOJeeHtwqR62hmUD0Ed1vruMdvq9jmdUNeWWoUH1">
    <button type="submit" class="btn btn-success">Submit</button>
</form>
```

Best Practices

Be careful with the database architecture, always use the proper length for specific column and never forget the indexes for searchable columns

```
Schema::create('article', function (Blueprint $table) {
    $table->increments( column: 'id');
    $table->string( column: 'title',  length: 200);
    $table->text( column: 'description')->nullable();
    $table->enum( column: 'status', ['active', 'inactive'])->defaults('active');
    $table->timestamps();
    $table->index(['title', 'status']);
});
```

Best Practices

Avoid the big query unless you really have to do it. The big query is hard to debug and understand.

```
->leftJoin('post', 'post.id', '=', 'notification.post_id')->whereNull('post.deleted_at')
->leftJoin('user', 'user.id', '=', 'post.user_id')
->leftJoin('post_comment', 'post_comment.id', '=', 'notification.comment_id')
->where(function ($query) use ($loggedUser) {
    $query->where('notification.type', '!=', 'video_completed')
        ->where('notification.by_user_id', '!=', $loggedUser->id)
        ->whereNotIn('notification.by_user_id', Magic::getBlockedUserIds())
        ->where(function ($query) use ($loggedUser) {
            $query->where(function ($query) use ($loggedUser) {
                $query->where('post.user_id', $loggedUser->id)
                    ->where(function ($query) use ($loggedUser) {
                        $query->where('notification.type', '!=', 'post_tag')
                            ->where('notification.type', '!=', 'comment_tag');
                    });
            })->orWhere(function ($query) use ($loggedUser) {
                $query->where('notification.user_id', $loggedUser->id)
                    ->where(function ($query) use ($loggedUser) {
                        $query->where('notification.type', 'post_tag')
                            ->orWhere('notification.type', 'comment_tag');
                    })
                    ->where('post.status', 'active')
                    ->where('post.completed', 'yes');
            })->orWhere(function ($query) use ($loggedUser) {
                $taggedInPosts = $loggedUser->taggedInPosts();
                $query->where(function ($query) use ($loggedUser) {
                    $query->where('notification.type', 'post_like')
                        ->orWhere('notification.type', 'post_comment');
                })
                    ->whereIn('notification.post_id', $taggedInPosts);
            });
        })
        ->whereNull('user.deleted_at');
})
->orWhere(function ($query) use ($loggedUser) {
    $query->where('notification.type', 'video_completed')
        ->where('notification.by_user_id', $loggedUser->id)
        ->whereNull('user.deleted_at');
})
```

Documentation

Documentation

https://laravel.com/docs/9.x

Starter kits

https://laravel.com/docs/9.x/starter-kits