**PHP 8 Features:-**

1. Union Types
2. Just In Time Compilation
3. Named Arguments
4. Match Expressions
5. Attributes
6. Constructor Property Promotion
7. Nullsafe Operator
8. Weak Maps
9. Saner string to number comparisons
10. Consistent type errors for internal functions

**LARAVEL**

**Laravel Install Command:-**

composer create-project laravel/laravel projectName

* Fillable – for mass assignment like insertion and updation. These fields are insertable.
* Guarded = remove the field for mass assignment like insertion and updation.

**MIDDEL WARE:-**

1 . **GLOBAL MIDDLEWARE**

It will run on every HTTP request of the application

2. **ROUTE MIDDLEWARE**

-the middleware can be registered at app/Http/Kernel.php.

-The middleware can be registered at app/Http/Kernel.php.

-This file contains two properties $middleware and $routeMiddleware.

-$middleware property is used to register Global Middleware and

-$routeMiddleware property is used to register route specific middleware.

Add the following line of code in app/Http/routes.php file in order to execute the middleware

Route::get( 'terminate', ['middleware' => 'terminate', 'uses' => 'ABCController@index',]);

**SERVICE CONTAINER:-**

It is a tools used to manage dependencies over the class and perform dependency injections.

It is used as a registry to keep track of all the classes in use within your application.

It also helps in binding interfaces to concrete classes.

Initiating object in the class create tight coupling, in order to avoid that we use service container to bind the object and we use it in the class.

Ref:- <https://www.codemag.com/Article/2212041/Dependency-Injection-and-Service-Container-in-Laravel>

**SERVICE PROVIDER:-**

* We have to create service provider by using the following command
* php artisan make:provider MyServiceProvider
* Then we have to register this provider in config/app.php file
* Add your class by using bind or singleton method in the register() function of the provider class

**EVENTS And LISTENERS:-**

1. Register the event and listener class in the file app/Providers/EventServiceProvider.php It is a key event pair

Eg:- LoginHistory::class => [StoreUserLoginHistory::class]

1. php artisan event:generate

It generate the listener class

1. Event file is created in app/Events folder

For example app/Events/LoginHistory.php file was created

1. Listener is created in app/Listeners folder

For example app/Listeners/storeUserLoginHistory.php file was created

1. Dispatch the event by the command for eg:- event(new LoginHistory($user));

Ref: https://dev.to/kingsconsult/laravel-8-events-and-listeners-with-practical-example-9m7

**SCHEDULERS:-**

* We have to schedule the command using app/Console/Kernel.php
* We can create command using php artisan make:command sendWelcomeEmail
* How to run the cron in production
* Access the server cron table by “crontab –e”
* Add the following line to the cron table: This will run the scheduler every minute.
* \* \* \* \* \* php /my-laravel-project/artisan schedule:run >> /dev/null 2>&1
* for local development and quick debugging, we can run the scheduler using the following command:
* php artisan schedule:work
* Ref <https://www.airplane.dev/blog/how-to-schedule-jobs-with-laravel-schedule>

**FACTORIES:-**

* Used to generate dummy data or test data
* Php artisan make : factory command to create a factory file in database / factories folder
* Every model created with the Artisan command make:model command by default uses the HasFactory trait which provides access to the factory() method
* php artisan make:factory UserFactory --model=User

**OBSERVER:-**

Ref

https://github.com/mobiosolutions/laravel-observer-example

https://www.itsolutionstuff.com/post/laravel-8-model-observers-tutorial-exampleexample.html

**TINKER**:-

* Php artisan tinker
* It is a command line tool to run php, laravel commands

**Model:-**

**Migration:-**

<https://kinsta.com/blog/laravel-model-factories/>

**QUERY SCOPE IN LARAVEL :-**

<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Post extends Model

{

public function scopePublished($query)

{

return $query->where('published', true);

}

public function scopeDraft($query)

{

return $query->where('published', false);

}

}

Call like this this

$publishedPosts = Post::published()->get();

$draftPosts = Post::draft()->get();

ACCESSOR AND MUTATORS:-

They are custom, user defined methods.

Accessors are used to format the attributes when you retrieve them from database.

Mutators are used to format the attributes before saving them into the database.

**LARAVEL OPTIMIZATION:-**

--------------------

1. CONFIG CACHES:-

laravel caches the config setting. If cached then modification of config will not affected.

php artisan config:cache //to cache the config settings

php artisan config:clear // clear the config caches

2. ROUTES CACHES:-

-------------

Route is an array. In order to cache the routes for speedy retrieval

php artisan route:cache //to cache the routes

php artisan route:clear // to clear the routes cache

3. view caching

event caching

application caching(memcache, redis)

\*. Remove Unused Service

remove unused service provider in config/app.php. Because unnecessarily loading unwanted service provider.

\*) Class Map OPTIMIZATION:-

Laravel calls multiple files to include requests, so even a mid-level Laravel app will have many files.

A simple trick is to declare all the included files to include requests and combine them in a single file.

Thus, a single file will be called and loaded for all include requests.

3. Avoid raw query

\*) Asset Optimization:- Combine all js and css files using laravel-mix feature

\*) Queue:- Use queue for very long running operations.

\*) Eager loading in orm to avoid N+1 query problem

// Before

$posts = Post::all();

foreach ($posts as $post) {

$comments = $post->comments;

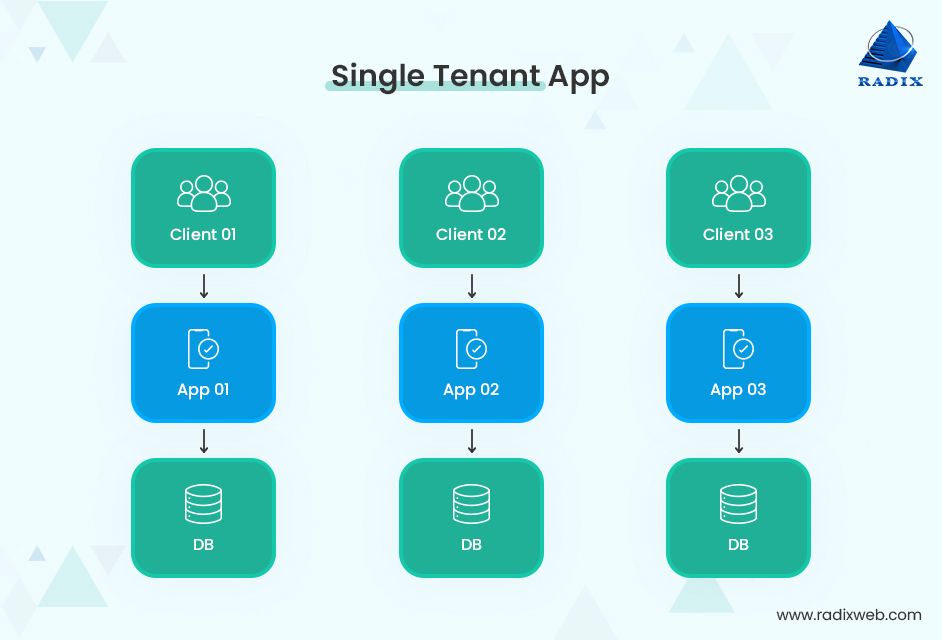
}

// After

$posts = Post::with('comments')->get();

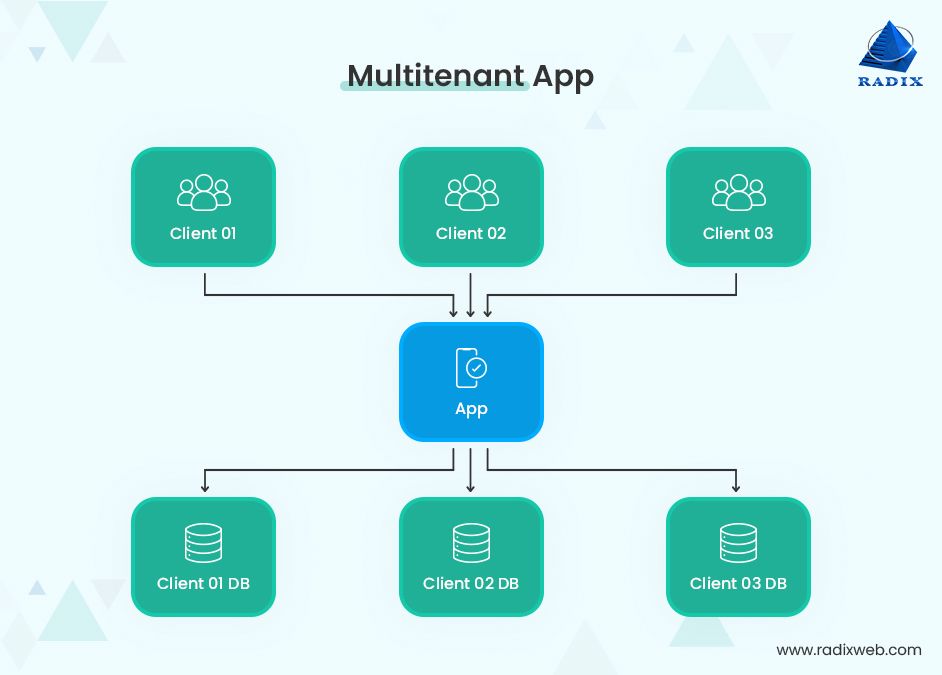
**OTHERS**

**SINGLE TENANT ARCHITECTURE:-**

* Separate DB and code for each instance for example x.abc.com, y.abc.com
* More expensive compare to multi tenant architecture
* 

**MULTI TENANT ARCHITECTURE:-**

* Each instance has common DB and code



Ref: https://radixweb.com/blog/saas-architecture

**REST ARCHITECTURE:-**

1. **Client-server architecture**. The sender and receiver are independent of each other regarding technology, platforming, programming language, and so on.

2. **Layered**. The server can have several intermediaries that work together to complete client requests, but they are invisible to the client.

3. **Uniform interface**. The API returns data in a standard format that is complete and fully useable.

4. **Stateless**. The API completes every new request independently of previous requests.

5. **Cacheable**. All API responses are cacheable.

6. **Code on demand**. The API response can include a code snippet if required.

**DESIGN PATTERN**

**Repository Design Pattern:-**

It is a structural pattern that separates the data access layer from the business logic layer**.**

For eg:- getAll, findById, create, update, and delete.

Login & Registration

1. https://www.itsolutionstuff.com/post/laravel-custom-login-and-registration-exampleexample.html

2. https://medium.com/@bbkgull/creation-of-user-login-and-registration-screens-with-laravel-8-8e563c7d5336

3. https://www.fundaofwebit.com/laravel-8/laravel-8-login-registration-system-tutorial