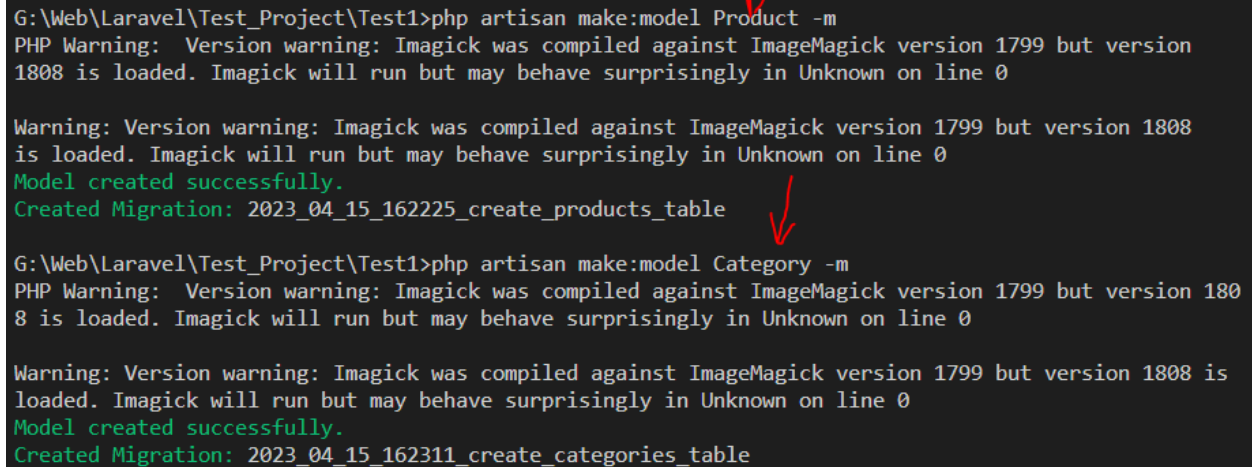


php artisan make:model Product -m

\*\*\*\*\*

php artisan make:model Category -m

\*\*\*\*\*



```
G:\Web\Laravel\Test_Project\Test1>php artisan make:model Product -m
PHP Warning:  Version warning: Imagick was compiled against ImageMagick version 1799 but version 1808 is loaded. Imagick will run but may behave surprisingly in Unknown on line 0

Warning: Version warning: Imagick was compiled against ImageMagick version 1799 but version 1808 is loaded. Imagick will run but may behave surprisingly in Unknown on line 0
Model created successfully.
Created Migration: 2023_04_15_162225_create_products_table

G:\Web\Laravel\Test_Project\Test1>php artisan make:model Category -m
PHP Warning:  Version warning: Imagick was compiled against ImageMagick version 1799 but version 1808 is loaded. Imagick will run but may behave surprisingly in Unknown on line 0

Warning: Version warning: Imagick was compiled against ImageMagick version 1799 but version 1808 is loaded. Imagick will run but may behave surprisingly in Unknown on line 0
Model created successfully.
Created Migration: 2023_04_15_162311_create_categories_table
```

\*\*\*\*\*

We should be attention to order of migration:

```
G:\Web\Laravel\Test_Project\Test1>php artisan migrate --path=database\migrations\2023_04_15_162311_create_categories_table.php
PHP Warning:  Version warning: Imagick was compiled against ImageMagick version 1799 but version 1808 is loaded. Imagick will run but may behave surprisingly in Unknown on line 0

Warning: Version warning: Imagick was compiled against ImageMagick version 1799 but version 1808 is loaded. Imagick will run but may behave surprisingly in Unknown on line 0
Migrating: 2023_04_15_162311_create_categories_table
Migrated: 2023_04_15_162311_create_categories_table (235.65ms)

G:\Web\Laravel\Test_Project\Test1>php artisan migrate
PHP Warning:  Version warning: Imagick was compiled against ImageMagick version 1799 but version 1808 is loaded. Imagick will run but may behave surprisingly in Unknown on line 0

Warning: Version warning: Imagick was compiled against ImageMagick version 1799 but version 1808 is loaded. Imagick will run but may behave surprisingly in Unknown on line 0
Migrating: 2023_04_15_162225_create_products_table
Migrated: 2023_04_15_162225_create_products_table (2,467.32ms)
```

\*\*\*\*\*

```
Schema::create('categories', function (Blueprint $table) {
    $table->id();

    $table->string("name", 50);
    $table->string("image", 150);

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

\*\*\*\*\*

```
Schema::create('products', function (Blueprint $table) {
    $table->id();

    $table->string("name",50);
    $table->string("image", 125);
    $table->unsignedBigInteger('category_id');

    $table->foreign("category_id")
        ->references("id")
        ->on("categories")
        ->onDelete('cascade');

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

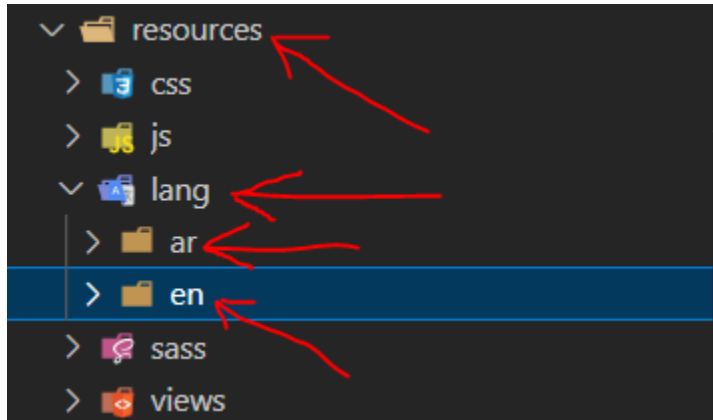
\*\*\*\*\*

php artisan make:controller API/ProductController

php artisan make:controller API/CategoryController

\*\*\*\*\*

**Note 1:** To define our languages file (translate file) we define first the language as folder in resources/lang



**Note 2:** to define our app language we use  
`app()->setLocale('language_here')`

**Ex:** `app()->setLocale('en')`

**Note 3:** to use our language file we use trans-function as  
`trans('file_name.attr_name');`

**Note 4:** To define the language of app we should use middleware.

\*\*\*\*\*

To define our middleware:

***php artisan make:middleware LanguageMiddleware***

It located in **app/Http/Middleware**.

```
public function handle(Request $request, Closure $next)
{
    $lang = $request->hasHeader("lang") ? $request-
>header('lang') : "en";
    app()->setLocale($lang);
    return $next($request);
}
```

To use it we should define it in **app/Http/Kernel.php** in **\$routeMiddleware**.

```
protected $routeMiddleware = [
    'auth' => \App\Http\Middleware\Authenticate::class,
    'auth.basic' =>
        \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class
    ,
    'cache.headers' =>
        \Illuminate\Http\Middleware\SetCacheHeaders::class,
    'can' =>
        \Illuminate\Auth\Middleware\Authorize::class,
    'guest' =>
        \App\Http\Middleware\RedirectIfAuthenticated::class,
    'password.confirm' =>
        \Illuminate\Auth\Middleware\RequirePassword::class,
    'signed' =>
        \Illuminate\Routing\Middleware\ValidateSignature::class,
    'throttle' =>
        \Illuminate\Routing\Middleware\ThrottleRequests::class,
    'verified' =>
        \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,
    'lang' =>
        \App\Http\Middleware\LanguageMiddleware::class,
];
```

\*\*\*\*\*

This for Testing the Language Middleware that we registered it previously.

```
Route::get('get-language', function() {
    echo app()->getLocale();
})->middleware('lang');
```

\*\*\*\*\*

**Note:** When we delete or update image content like Category we should set the delete or update the image content in the last step.

**Note 2:** The *\$category->has* && *\$category->filled* has the same way of manipulate the text data.

**Note 3:** to check if the request has file we use *\$request->hasFile('The\_File\_Name')*:

```
if($request->hasFile('image')) {  
    // Delete The Old Image.  
    unlink(storage_path('/app/public/images/categories/' . "Test 3" .  
"/". $category->image));  
    // set the new one.  
}
```

\*\*\*\*\*

**Note 1:** When we run background job we must run: *php artisan queue:work*

**Note 2:** We must be attention when we modify the code and the background work is running we must restart it for make changes.

**Note 3:** If we set **queue:listen** then we listen for events and works.

\*\*\*\*\*

**Note 1:** To make an event we can use: *php artisan make:event EventName*.

**Note 2:** To Listen to event we can use: *php artisan make:listener* && we can make it inherit from `ShouldQueue` and use: `use InteractsWithQueue`

```
class ProductEventListener implements ShouldQueue
{
    use InteractsWithQueue;
    ...
}
```

\*\*\*\*\*

**Note 1:** To Retry the failed jobs: *php artisan queue:retry id*

**Note 2:** To Retry All failed jobs: *php artisan queue:retry all*

\*\*\*\*\*

If we want to delete an object as Images from storage using **queues || background Jobs || events**, we should set the parameter as the image name (as String) (Not required with **SerializesModels**).

**Note 1:** If we want to pass an Model Object as parameter for Background Job we should set as args for Job the meta-data of Model Object (id, unique name, unique email, ...etc).

**Note 2:** If we forget to run *php artisan queue:work* and we send Model Object as parameter then it will lose its data.

**Note 3:** The Solution of above problem by make the model using:

```
// after declaring the namespace of model.  
use Illuminate\Queue\SerializesModels;  
  
// in the body of Model class.  
use HasFactory, SerializesModels;
```

\*\*\*\*\*

**Note 1:** The Schedule of Laravel Only Run the command in specific time only once per one run.

**Note 2:** To Run the Schedule on periodic times; for Laravel 8:

*php artisan schedule:work.*

\*\*\*\*\*



To Create Our New Queue Connection; open **config/queue.php** and define our connection on connection-array:

```
'connections' => [
    ...
    'product_connection' => [
        'driver' => 'database',
        // This is the connection of database.
        // 'connection' => 'default',

        // this must be set for defining the table of jobs
        'table' => 'jobs',
        'queue' => 'products',
        'retry_after' => 90,
        'block_for' => null,
    ],
    ...
]
```

To Config my dispatch to work on that queue we use

**onQueue('queue-name-here')-method:**

```
deleteProductImageJob::dispatch($product)->onQueue('products');
```

To Run The Jobs on Product-Queue we use:

***php artisan queue:work --queue=products***

To Run the Jobs on different Connection of Queues we use

**onConnection('connection-name-here')-method:**

```
deleteProductImageJob::dispatch($product)->onConnection('product_connection');
```

To run the jobs on that connection:

***php artisan queue:work product\_connection***

\*\*\*\*\*

To make specific event run on specific queue we set the:

```
// define the queue that should run on it.  
public $queue = 'products';
```

In The Handler class that extends:

```
class ProductEventListener implements ShouldQueue  
{  
    use InteractsWithQueue, Queueable, SerializesModels, Dispatchable;  
    ...  
}
```

This will make the Job run on *products-queue* and saved *in db in products queue*.

\*\*\*\*\*

**Note 1:** The Search in Laravel 8 is case insensitive.

Note 2: To Make Scope Query For Specific Model:

1. Define the method starts with scope-keyword.

```
public function scopeName($query) {  
    return $query->where('name', 'LIKE', '%TEST%');  
}
```

2. Use it with model by calling it without scope-keyword.

```
$categories = Category::with(['products' => function($q){  
    $q->name();  
}])->get();
```

```
public function scopeDynamicName($query, $name) {  
    return $query->where('name', 'LIKE', "%$name%");  
}
```

```
$categories = Category::dynamicName($name)->get();
```

\*\*\*\*\*

To use Redis With Laravel 8:

- Composer require predis/predis;
- Add the line to .env-file:
  - REDIS\_CLIENT=predis

```
# This for connection with redis.  
REDIS_HOST=127.0.0.1  
REDIS_PASSWORD=null  
REDIS_PORT=6379  
REDIS_CLIENT=predis
```

\*\*\*\*\*

The Redis Prefix our value by specific value:

Note 1: comment the line of prefix to get the right values for keys, when using keys-method of redis-package (predis-package).

```
'redis' => [  
  
    'client' => env('REDIS_CLIENT', 'phpredis'),  
  
    'options' => [  
        'cluster' => env('REDIS_CLUSTER', 'redis'),  
        // 'prefix' => env('REDIS_PREFIX', Str::slug(env('APP_NAME',  
'laravel'), '_').'_database_'),  
    ],  
],
```

\*\*\*\*\*

```
# This for connection with redis.  
REDIS_HOST=127.0.0.1  
REDIS_PASSWORD=null  
REDIS_PORT=6379  
REDIS_CLIENT=predis  
# This will load the database 2.  
REDIS_DB=2
```

\*\*\*\*\*

