BASIC LARAVEL

BY AKMAL

ISI KANDUNGAN

1) SETUP LARAVEL

A. INSTALL SOURCE TREE

1. Download Source Tree : Sourcetree | Free Git GUI for Mac and Windows (sourcetreeapp.com)

2. Install

B. INSTALL LARAGON

1. Download from website : https://laragon.org/download/index.html

2. Install

3. Quick Add

- tools -> quick_add -> phpMyAdmin

4. Open Terminal and run

- php -v

(if everything doesn't have error then u r done installing laragon otherwise go install php first)

* INSTALL PHP *

1. download (https://windows.php.net/download#php-8.2)

2. extract and put in (C:/laragon/bin/php/php-8.2.1-Win32-vs16-x64)

C. INSTALL COMPOSER

1. Download from website : https://getcomposer.org/Composer-Setup.exe

2. Install Composer in laragon (laragon/bin/php/php.exe)

3. Choose php latest (right click ->php)

4. Open Terminal in laragon and run

- composer --version

D. SETUP SOURCE TREE AND GITLAB (https://youtu.be/dLRZyn6PapU)

1. Clone GitLab to SourceTree

- https://gitlab.com/mnhazim/api-resitku.git

- C:\laragon\www\api.resitku

- api.resitku

- ROOT

E. SETUP DATABASE

1. Edit config.sample.inc.php (in"C:\laragon\etc\apps\phpMyAdmin")

- AllowNoPassword = true

- save

2. Laragon Setting -> service and ports -> port = 3307

3. Run on browser :

http://localhost/phpmyadmin/

- username : root

- pw : biarkan kosong

4. Import sql file

- create new db name : api.resitku

- import sql file

F. SETUP LARAVEL

1. Run on terminal vs code

enable sodium in php.ini (just remove the semicolumn bfr word sodium)

composer install / composer update

copy .env.example .env

php artisan key:generate

php artisan passport:install

php artisan config:cache

2.restart laragon

3. php artisan serve

4. login

* INSTALL LARAVEL *

1) Open terminal laragon

2) Run = composer create-project laravel/laravel example-app

3) Run = php artisan –version

LEARN LARAVEL
https://youtube.com/playlist?list=PLnrs9DcLyeJTG-_mjD68Gn0sC5hbzaU2T

2) ARTISAN COMMAND

php artisan make:model User -mcr -R / migration , controller , resourcse , Request

php artisan make:model User

php artisan make:controller UserController --resource –model=User

php artisan make:migration create nameOfTableInPlural(s) _table

php artisan optimize

php artisan route:cache

php artisan route:clear

php artisan config:cache

php artisan migrate:status

php artisan migrate:rollback –step=2

Ternary Operator

expression ? true_value : false_value

{{ $user -> is_active == 1 ? 'active' : 'inactive' }}

Null coalesces

$variable = $x ?? $y;

Formatting

1. Number

number_format($investorDashboard['sumIncome'], 2)

2. Date

date_created->format('d-m-Y')

3. Time

{{ date("d-m-y", strtotime($item->time) }} //16-12-2022

{{ date("H:i:s", strtotime($item->time)}} //10:39:00

$date = Carbon\Carbon::parse($inputHere)->format('d-m-Y'); //16-12-2022

$time = Carbon\Carbon::parse($inputHere)->format('H:i A');}} //10:39 AM

Character	Meaning	Example
d	day of the month with leading zeros	03 or 17
j	day of the month without leading zeros	3 or 17
D	day of the week as a three-letter abbreviation	Mon
l	full day of the week	Monday
m	month as a number with leading zeros	09 or 12
n	month as a number without leading zeros	9 or 12
M	month as a three-letter abbreviation	Sep
F	full month	September
y	two-digit year	18
Y	full year	2018

Character	Meaning	Example
d	day of the month with leading zeros	03 or 17
j	day of the month without leading zeros	3 or 17
D	day of the week as a three-letter abbreviation	Mon
l	full day of the week	Monday
m	month as a number with leading zeros	09 or 12
n	month as a number without leading zeros	9 or 12
M	month as a three-letter abbreviation	Sep
F	full month	September
y	two-digit year	18
Y	full year	2018

3) ARRAY, COLLECTION & HELPER

ARRAY

1. Foreach

```
foreach ($expenses as $key => $expense) {                    //$key = index number
    $income = $incomes[$key];
     $profitMargin[] = $income == 0 ? 0 : (($income - $expense) / $income) * 100;
     $roi[] = $expense == 0 ? 0 : (($income - $expense) / $expense) * 100;
}
```

`$loop->index`	The index of the current loop iteration (starts at 0).
`$loop->iteration`	The current loop iteration (starts at 1).
`$loop->remaining`	The iterations remaining in the loop.
`$loop->count`	The total number of items in the array being iterated.
`$loop->first`	Whether this is the first iteration through the loop.
`$loop->last`	Whether this is the last iteration through the loop.
`$loop->even`	Whether this is an even iteration through the loop.
`$loop->odd`	Whether this is an odd iteration through the loop.
`$loop->depth`	The nesting level of the current loop.
`$loop->parent`	When in a nested loop, the parent's loop variable.

2. Forelse

```
@forelse ($expenses as $key => $expense)
        $income = $incomes[$key];
         $profitMargin[] = $income == 0 ? 0 : (($income - $expense) / $income) * 100;
        $roi[] = $expense == 0 ? 0 : (($income - $expense) / $expense) * 100;
@empty
        “Data is empty”
@endforelse
```

3. Array_sum (to sum data in array)

```
$countIncome = array_sum($this->chartRepository >countIncomeForConsumerOverall($consumerIds));
```

4. In Array

```
$array = [1, 2, 3, 4, 5];

if (in_array(3, $array)) {
   echo "3 is in the array!";
```

5. Array Chunk

```
foreach ( array_chunk ($user_data , 1000) as $item)
{
        Student::insert($item);
}
```

6. Array Fill

```
$value = array_fill( $start_index, $size_of_array, $data )
```

COLLECTION / HELPER

1) **Map / Loop**

```
$squares = collect([1, 2, 3, 4, 5]) ->map(function ($n) {                    // foreach($numbers as $n)
        return $n * $n;
});                                                                          //Output @ squares : [1, 4, 9, 16, 25]
```

```
User::where('investor_id',$investor->id)->each(function(User $user){
    $user->investor_id = null;
    $user->save();
});
```

2) **Simple method**

```
$numbers = [1, 2, 3, 4, 5];
$users = User::all();
```

```
$average = collect($numbers)->avg();                $max = collect([1, 2, 3, 4, 5])->max();
$names = $users->pluck('name');                     $median = collect([1, 1, 2, 4])->median();
$user = $user->count();                             $min = collect([1, 2, 3, 4, 5])->min();
```

```
$collection->push(5);                               $collection->reverse()
$collection->pop();                                 collect([])->isEmpty();
$collection->random();                              collect([])->isNotEmpty();
$collection->shuffle();
```

```
$collection = collect([1, 2, 3, 4, 5, 6, 7]);       $collection = collect([1, 2, 3, 4, 5]);
$chunks = $collection->chunk(4);                    $diff = $collection->diff([2, 4, 6, 8]);
// [[1, 2, 3, 4], [5, 6, 7]]                        // [1, 3, 5]
```

```
$collection = collect(['Desk', 'Sofa', 'Chair']);
$intersect = $collection->intersect(['Desk', 'Chair', 'Bookcase']);          ->duplicates();
```

```
->toJson()              @           ->take()
->sortBy('price')       @           ->sortByDesc('price')
```

4) LAYOUT AND COMPONENT

4.LAYOUT

layout -> app.blade.php

```
<div>
        @stack('style')

        <title> @yield ( 'title' ) </title>

        . . . . . .

        @yield ('content')

        @stack('script')
</div>
```

post -> index.blade.php

```
<div>
        @extends ( layout.app )

        @push('style')
                <link rel="stylesheet" type="text/css" href="{{ asset('theme/css/chart-apex.css') }}">
        @endpush

        @section ('title' , 'Home Page')

        @section ('content')
        . . . . . .

        . . . . . some code here . . . . . .. . . .

        @endsection

        @push('script')
                <script src="{{ asset('theme/vendors/js/charts/apexcharts.min.js') }}"></script>
        @endpush

</div>
```

COMPONENT

1. php artisan make:component Alert

2. In app -> view/component/ Alert.php

```
public $message;

public function __construct ( $message )
{
        $this->message = $message;
}
```

3. In resources -> view -> component -> alert.blade.php

```
<div>
        . . . . . .
        . . . . . . . . . some code here . . . . . .. . . .
        . . . . . .
        . . . . . . . . . some code here . . . . . .. . . .
        {{          $message          }}
</div>
```

4. Use anywhere in Blade

```
<x-alert message = ' hello world ' >
        . . . . . .
        . . . . . . . . . some code here . . . . . .. . . .
        . . . . . .
        . . . . . . . . . some code here . . . . . .. . . .
</x>
```

5) ROUTE

~ php artisan route:list

Type of route

1.Obtain data -> Route::get($uri, $callback);

2. Create data -> Route::post($uri, $callback);

3. Update data -> Route::put($uri, $callback);

4. Delete data -> Route::delete($uri, $callback);

1. Route View

a) Route::view('/welcome', 'welcome');

b) Route::view('/welcome', 'welcome', ['name' => 'Taylor']);

2. Route Get / Post

a) Route::get('/greeting', function () {

return 'Hello World'; @ return view('contact');

return view('pages.expense-category.index', compact('expenseType'));

});

b) Route::post('email', [FeedbackController::class, 'submit'])->name('email.submit');

c) Route::redirect('/contact', '/contact-us');

3. Route Resource

use App\Http\Controllers\PhotoController;

Route::resource('photos', PhotoController::class);

Verb	URI	Action	Route Name
GET	/photos	index	photos.index
GET	/photos/create	create	photos.create
POST	/photos	store	photos.store
GET	/photos/{photo}	show	photos.show
GET	/photos/{photo}/edit	edit	photos.edit
PUT/PATCH	/photos/{photo}	update	photos.update
DELETE	/photos/{photo}	destroy	photos.destroy

6) MIGRATION

```
  public function up()
  {
    Schema::create('users', function (Blueprint $table) {

        // creates a big integer column that will be used as the primary key of the table
        $table->bigIncrements('id');


        // Required means that the column cannot be null and a value must be provided
        $table->string('name')->required();


        // unique means the column cannot contain duplicate values  when inserting data into the table
        $table->string('email')->unique();


        // nullable means can contain null values
        $table->timestamp('email_verified_at')->nullable();


         // unsignedBigInteger means big integers values and will only accept non-negative numbers
        // creates a foreign key on the column "role_id" that references the primary key of the "roles" table

        $table->unsignedBigInteger('role_id');
        $table->foreign('role_id')->references('id')->on('roles');      @           $table->foreignId('user_id')->constrained();


       // This method creates two columns named "created_at" and "updated_at"
        $table->timestamps();

    });
  }


  public function down()
  {
    // Drops the "users" table if it exists

    Schema::dropIfExists('users');

  }
}
```

TYPE OF DATA FOR MIGRATION

1. **Number** (float -> decimal -> double)

```
// ensures that the column can only contain non-negative integers
$table->unsignedInteger('age');

// small integer type that can store values from 0 to 255 + can only contain non-negative integers
$table->unsignedTinyInteger('is_active')->default(0);

// no decimal place + can be +ve or -ve number + can store up to 2147483647
$table->integer('votes')

// integer + store a value up to 9223372036854775807.
$table->bigInteger('votes')

// small-sized integer type that can store values from -32768 to 32767
$table->smallInteger('rating');

// medium-sized integer type that can store values from -8388608 to 8388607
$table->mediumInteger('num_of_orders');


//don't require a specific precision and are okay with some rounding errors
$table->float('amount', 8, 2);

// can hold up 8 digits in total and 2 digits after the decimal point. ( price @ anything yang tak mementingkan ketepatan)
$table->decimal('price', 8, 2);

// can hold up 8 digits in total and 6 digits after the decimal point ( lebih ketepatan dlm menjangka data)
$table->double('latitude', 8, 6);

$table->boolean('is_admin')->default(false);                    //used to store a boolean value (true or false)
```

2. String

```
// store a string of variable length, up to 255 characters ( name / short description )
$table->string('email')->unique();

// store a longer string of text, up to 65,535 characters ( product description / user bio )
$table->text('description');


// can store up to 16777215 characters.                         // store a very long string
$table->mediumText('description')                               $table->longText('notes']


// This method creates a char column named "gender" with a length of 1 ("M" or "F)
$table->char('gender', 1);
```

3. Time

```
$table->dateTime('start_date');                                 //store date and time values.

$table->time(): creates a time column                           $table->timestamp(): creates a timestamp column

$table->date(): creates a date column                           $table->year(): creates a year column
```

7) MODEL

```
<?php

namespace App\Models;


class AccountingService extends Model

{

   //properties model

   protected $table = ‘products’;                                    //define table name

   protected $primaryKey = ‘akmal_id’;

   protected $incrementing = false;

   protected $keyType = ‘string’;


   public $timestamps = false;

   protected $dateFormat = ‘Y-m-d’;

   public const CREATED_AT = ‘akmal_created’

   public const UPDATED_AT = ‘akmal_updated


   protected $fillable = [ 'user_id', 'status' ];                    //determine the column that want to be stored

   protected $hidden = 'status' ;                                    //hide the column query


   protected $casts = [

            'status'  => ‘int’                                       //stated type of some attribute

   ];


 public function posts()
{
             return $this->hasMany(Post::class)->where(‘status’,’published’);
}


   // Define the accessor for the "name" attribute                  // Define the mutator for the "password" attribute

   public function getNameAttribute($value)                          public function setPasswordAttribute($value)

   {                                                                 {

            return ucfirst($value);                                         $this->attributes['password'] = bcrypt($value);

   }                                                                 }
```

8) RELATIONSHIP

1-1 TO RELATIONSHIP (1 "User" only have 1 "Profile")

```
//table that less use should have foreign id
class User extends Model       //IN USER MODEL
{
    public function profile()
    {
        return $this->hasOne(Profile::class);
    }
}
```

```
class Profile extends Model    //IN PROFILE MODEL
{
    public function user()
    {
        return $this->belongsTo(User::class);
    }
}
```

1-TO-MANY (1 "User" have many "Post")

```
// foreign key should be in many table
class User extends Model       //IN USER MODEL
{
     public function posts()
     {
            return $this->hasMany(Post::class);
      }
}
```

```
class Post extends Model       //IN POST MODEL
{
    public function user()
    {
        return $this->belongsTo(User::class);
     }
}
```

MANY TO MANY

//foreign key should have in bridge/pivot table

1. Make A Bridge Table migration **//user -> role_user <- role**

php artisan make:migration create_role_user_table

2. In migration Class

```
public function up()
{
   Schema::create('role_user', function (Blueprint $table) {
      $table->unsignedBigInteger('role_id');
      $table->unsignedBigInteger('user_id');
      $table->foreign('role_id')->references('id')->on('roles');
      $table->foreign('user_id')->references('id')->on('users');
   });
}
```

3. In Model

```
// app/Models/User.php
class User extends Authenticatable
{
   public function roles()
   {
      return $this->belongsToMany(Role::class, 'users_roles');
   }
}
```

```
// app/Models/Role.php
class Role extends Model
{
   public function users()
   {
      return $this->belongsToMany(User::class, 'users_roles');
   }
}
```

9) POLYMORPHIC

A) 1 TO 1 / 1 TO MANY (Comment model and a Post or Video model)

1. Create migration

```
php artisan make:model Comment -m                    //create migration and model
```

```
Schema::create('comments', function (Blueprint $table) {
    $table->id();
    $table->text('body');
    $table->unsignedBigInteger('commentable_id');
    $table->string('commentable_type');
    $table->timestamps();
});
```

2. Do relations in comment model

```
class Comment extends Model
{
    public function commentable()
    {
            return $this->morphTo();
    }
}
```

3. In other class

```
class Post extends Model
{
    public function comment()
    {
      return $this->morphOne(Comment::class, 'commentable');
                        @
                morphMany()
     }
}
```

```
class Video extends Model
{
    public function comment()
    {
      return $this->morphOne(Comment::class, 'commentable');
                        @
                morphMany
     }
}
```

4. To call it just do this

```
$post = Post::find(1);
$comment = $post->comment()->create(['body' => 'This is a comment.']);
```

B) MANY TO MANY relationship (Post and Tag, and we want to define a relationship where a post can have many tags, and a tag can be associated with many posts.)

1. Create a migration

(normal table)

```
php artisan make:migration create_posts_table --create=posts
```

```
Schema::create('posts', function (Blueprint $table) {
    $table->id();
    $table->string('title');
    $table->text('body');
    $table->timestamps();
});
```

```
php artisan make:migration create_tags_table --create=tags
```

```
Schema::create('tags', function (Blueprint $table) {
    $table->id();
    $table->string('name');
    $table->timestamps();
});
```

(many to many table)

```
php artisan make:migration create_post_tag_table --create=post_tag
```

```
Schema::create('post_tag', function (Blueprint $table) {
    $table->id();
    $table->unsignedBigInteger('post_id');
    $table->unsignedBigInteger('taggable_id');
    $table->string('taggable_type');
    $table->timestamps();
});
```

2. Create relation

```
class Post extends Model
{
    public function tags()
    {
        return $this->morphToMany(Tag::class, 'taggable');
    }
}
```

```
class Tag extends Model
{
    public function posts()
    {
        return $this->morphedByMany(Post::class, 'taggable');
    }
}
```

```
$post = Post::find(1);
```

```
$tag = new Tag(['name' => 'Technology']);
```

```
$post->tags()->save($tag);
```

10) ELOQUENT, CRUD &PAGINATION

1. VIEW

```
$student = Student :: all( [ 'id', 'email' ] ) -> orderBy('name', 'desc')->get();

$student = Student::where ( 'username', 'like' , ' %. $name .% ' ) ->get()  @ ->first()   @ ->last()  @ ->firstOrFail  @ ->toSql();

$post = Post::oldest()->get()                                   $post = Post::latest()->get()

$consumers = ConsumerDetail::with('incomes', 'expenses')->get();    $user = User::has('consumer')->get();


$consumers = ConsumerDetail::withCount('incomes')->get();          //withMin @ withMax @ withAvg  -loadwithrelation

{{ $consumer->incomes_count }}


$consumers = ConsumerDetail::loadCount('incomes')->get();          //loadMin @ loadMax @ loadAvg -loadonlycount

{{ $consumer->relation_count[incomes] }}



//others

-> firstOrCreate([put condition here],[put the data here])    @    -> updateOrCreate( [ ] , [ ] )

-> isEmpty()                                  ->isNotEmpty()                              ->wasRecentlyCreated()


email' => $request->input('official_email')          @          $request->official_email
```

2. ADD

```
public function featureCreate(Request $request, Plan $plan)
{
    $plan->features()->create($request->all());
            @
    $user = User::create ([
            'name' => $request->name,
            'email' => 'janedoe@example.com',
            'password' => bcrypt('password')
    ]);
    return redirect()->route('someURL');
}
```

3. UPDATE

```
public function update(Request $request, Plan $plan)
{
        $plan->update( $request->only(['name', 'value' ]) );
        $plan ->update( $request->all() ) ;
        $plan->update([
            'name' => $request->name,
        ]);
        $plan->update($request->all() + [
                        'is_active' => $request->has('is_active')
                    ]);
    return to_route('subscription.plan');
}
```

4. DELETE

```
public function featureDelete(PlanFeature $feature)
{
    $feature->delete();                                                    //soft delete
     @
     DB :: table('users')->where('email',$investor->official_email)->delete()      //permanent delete
    return redirect()->back();
 }
```

5. RESTORE (soft delete)

```
public function restore($featureid)
{
    $feature = Feature::withTrashed()->find($featureid);

    if ( $feature && $feature->trashed() )
            $feature->restore();

    return redirect()->back();
 }
```

6.UPSERT METHOD

```
$data = [
  [                                    [                                    [
    'name' => 'John Doe',                'name' => 'Jane Doe',                'name' => 'Bob Smith',
    'email' => 'john@example.com',       'email' => 'jane@example.com',       'email' => 'bob@example.com',
    'age' => 30,                         'age' => 25,                         'age' => 35,
  ],                                   ],                                   ],
];

$uniqueKey = ['email'];                  // Define the unique key for the records
$updateValues = ['age'];                 // Define the values to be updated if the records already exist
User::upsert($data, $uniqueKey, $updateValues);   // Use upsert to insert or update the records
```

WHERE CONDITION

1. **whereHas** : allows you to query for records based on the existence of related records

```
$users = User::whereHas('posts', function ($query) {
   $query->where('title', 'like', '%laravel%');
})->get();
```

2. **Or Where** : to check multiple conditions and return the results if any of them are true.

```
$users = User::where('votes', '>', 100)->orWhere('name', 'John')->get();
```

3. **Where in**: This is used to check if a column value exists in a given array of values.

```
$users = User::whereIn('id', [1, 2, 3])->get();   @  $users = User::find([1, 2, 3])->get();
```

4. **Where between**: This is used to check if a column value exists between two given values

```
$users = User::whereBetween('votes', [1, 100])->get();
$expenses = Expense::whereBetween('expense_date', [$start_date, $end_date])->get();
```

5. **WhereKey**

```
$users = User::whereNull('updated_at')->get();
$users = User::whereNotNull('updated_at')->get();
$users = User::whereName(‘akmal’)->get();
$users = User::whereEmail(‘akmal875@gmail.com’)->get();
```

6. **Where date**: This is used to check if a column is a specific date.

```
$users = User::whereDate('created_at', '2022-01-01')->get();
$users = User::whereDay('created_at', '01')->get();
$users = User::whereMonth('created_at', '01')->get();
$users = User::whereYear('created_at', 2022)->get()
```

6. **Where Relation**

```
$invoiceBill = PlanInvoice::->whereRelation('PlanSubscriber','subscriber_id',$subscriber_id)
                                                    model          | any key |      $value
```

7. **Using RAW and PLUCK**

```
 $incomes = Income::whereIn('consumer_detail_id', $consumerIds)
          ->where('income_status_id', 2)
         ->selectRaw('consumer_detail_id, count(*) as total, , sum(amount) as income_sum'')
         ->pluck('total', 'consumer_detail_id' 'income_sum',)
         ->toArray();
```

8. **When** : same like where but for a condition that doenst have involvement with database

```
$name = 'John';
$post = $user->posts();
        ->when($name, function ($query, $name) {
                return $query->where('name', $name);
            })
        ->get();
```

9. **Wrap** where and orWhere

```
$users = DB::table('users')
        ->when($request->filled('search'), function ($query) use ($request) {
            $query->where(function ($query) use ($request) {
                $query->where('name', 'like', '%' . $request->input('search') . '%')
                    ->orWhere('email', 'like', '%' . $request->input('search') . '%');
            });
        })
        ->get();
```

10. **First Or**

```
$user = User::where('name', 'John')->where('age', 30)
        ->firstOr(function () {
            return new User([
                'name' => 'Guest User',
                'age' => 18
            ]);
        });
```

11. **Latest of many** : function to sort based on the latest data created but on model page (in model class)

```
class Post extends Model
{
   public function comments()
  {
        return $this->hasMany(Comment::class);
   }

   public function latestComment()
   {
         return $this->hasOne(Comment::class)->latestOfMany();
    }
}
```

```
//In controller
$post = Post::find(1);
$latestComment = $post->latestComment;
```

Pagination

```
$post = Post::paginate(5, ['*'] , 'posts');          //$post->links()
$user = User::paginate(5, ['*'] , 'users');                    //$user->links()
```

11) REQUEST VALIDATION & FLASH MESSAGE

VALIDATION

1. Create new request

php artisan make:request CreateMaterialRequest

2. Function inside CreateMaterialRequest

```
public function rules()
{
   return [
      'name' => 'required | string | max:255',
      'email' => 'required | email | unique : users , email',
      'password' => 'required | string | min:8 | confirmed',
            @     'password' => Password::min(8)->letters()->numbers()->mixedCase()->symbols(),
      'age' => 'required |numeric |min:18',
      'gender' => 'required | string',
   ];
}
```

3. In Controller

```
public function create ( CreateMaterialRequest $request )
{
            Material :: create( $request->validated() );
            return redirect() -> back();
}
```

4. Apply in Blade

```
<form class="auth-login-form mt-2" method="POST" action="{{ route('login') }}">
            @if ( $errors->any() )
                        <div class=”alert alert-danger’>
                                    <ul>
                                                @foreach
                                                            <li>{{ @error }}</li>
                                                @endforeach
                                    </ul>
                        </div>
            @endif
            <input type="text" class="form-control" id="email" name="email" />
</form>
```

5) Here are some examples of common validation rules you can use:

- **required**: The field must be present in the input data and not empty.
- **string**: The field must be a string.
- **email**: The field must be a valid email address.
- **numeric**: The field must be a number.
- **date**: The field must be a valid date.
- **min:value**: The field must be at least **value** characters long.
- **max:value**: The field must be no longer than **value** characters.
- **unique:table,column**: The field must be unique in the specified database table and column
- **confirmed :** Laravel automatically checks if the two input fields are the same by taking the input value of the password_confirmation field and check if it matches the input value of the password field.

  <input type="password" name="password">

  <input type="password" name="password_confirmation">

FLASH MESSAGE

1) In Controller

```
public function add ( Request $request)
{
        $student = Student :: create;
        $student->create($request->all());

        if($student)
        {
                session :: flash (‘status’ , ‘success’);
                session :: flash (‘message’,’add successfully’);
        }

        return redirect(‘someURL’);

}
```

2) In Blade

```
@if(Session :: has (‘status’) )
        <div class=”alert” role=”alert” > {{ Session::get(‘message’) }} </div>
@endif
```

12) SEEDER & FAKER

1) CREATE A SEEDER FILE
php artisan make:seeder TableNameSeeder

2) IN SEEDER FILE

```
use App\Models\Plan;

use Faker\Factory as Faker;

public function run ( )
{
        //if you want to truncate ( delete all data in table ) + use schema
        schema      :: diasableForeignKeyConstraint();
        tableName  :: truncate();
        schema      ::enableForeignKeyContsraint();


        $faker = Faker::create();

        for ($i = 0; $i < 10; $i++) {

                DB::table('table_name')->insert([

                        'name' => $faker->name,

                        'email' => $faker->unique()->safeEmail,

                        'password' => bcrypt('password'),

                        'created_at' => now(),

                        'updated_at' => now(),

                ]);

        }

}
```

3) SEED ALONE
php artisan db :: seed –class=NameSeeder

4) SEED ALL

a) Class DatabaseSeeder

```
public function run()
{
        $this -> call ([

                UserSeeder :: Class

                PostSeeder :: Class

        )]
}
```

b) seed
php artisan migrate --seed

13) EMAIL

1. php artisan make:mail NewUserWelcomeMail --markdown=emails.new_user_welcome

This command will generate a new mail class at **App/Mail/NewUserWelcomeMail.php** and a new markdown template view at **resources/views/emails/new_user_welcome.blade.php**

2. IN MAIL CLASS

```
<?php

namespace App\Mail;


use Illuminate\Bus\Queueable;

use Illuminate\Mail\Mailable;

use Illuminate\Queue\SerializesModels;


class NewUserWelcomeMail extends Mailable

{

   use Queueable, SerializesModels;


   public $user;


   public function __construct($user)

   {

      $this->user = $user;

   }


   public function build()

   {

      return $this->markdown('emails.new_user_welcome')

                  ->subject('Welcome to My Site')

                  ->with([

                     'username' => $this->user->name,

                     'email' => $this->user->email,

                  ]);

   }

}
```

3. EDIT EMAIL TEMPLATE IN ('emails.new_user_welcome')

```
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>
<body>
  <p>
    We would like to inform you that your request has been received and successfully submitted. <br>
    Our team is currently reviewing it and will contact you as soon as possible.<br>
    We appreciate your patience and understanding, and kindly request that you refrain from submitting duplicate requests as this may delay our response time. <br>
    Thank you for choosing our service.
  </p>
</body>
</html>
```

4. CALL IN CONTROLLER

```
$user = User::find(1);
Mail::to($user->email)->send(new NewUserWelcomeMail($user))->queue();
```

14) REPOSITORY & TRAITS

REPOSITORY

1. **Create REPO** in app -> http -> repositories (ConsumerRepository.php)

```
namespace App\Repositories;

class ConsumerRepository extends BaseRepository {

        //can write any functions here

}
```

2) USE REPO IN ANY FUNCTION

```
use App\Repositories\ConsumerlRepository;

private $consumerRepository;

public function __construct(ConsumerRepository $consumerRepository)

{

    $this->consumerRepository = $consumerRepository;

}
```

```
public function index ()

{

    $action =  $this->consumerRepository->method_in_that_repo();
}
```

TRAITS

1. **CREATE A TRAITS** in => app/Traits/HasFullName.php

```
namespace App\Traits;

trait HasFullName

{

  public function getFullName()
  {
    return $this->first_name . ' ' . $this->last_name;
  }

}
```

2. **IN MODEL YOU NEED TO ADD THE TRAITS**

```
use App\Traits\HasFullName;          //user model

class User extends Authenticatable
{
    use Notifiable, HasFullName;
}
```

3. JUST **CALL THE FUNCTION** IN TRAITS LIKE YOU CALL THE RELATIONSHIP IN MODEL IN ANY CONTROLLER

```
$user = User::find(1);

$fullName = $user->getFullName();
```

15) JOB

1. php artisan make:job UpdateUserProfileJob

2. FUNCTION IN JOB

```
<?php
namespace App\Jobs;
use App\Models\User;

class UpdateUserProfileJob
{
   protected $user ,  $data;
   public function __construct(User $user, array $data)
   {
      $this->user = $user;
      $this->data = $data;
   }

   public function handle()
   {
      $this->user->update($this->data);
   }
}
```

3. USE IT ANY CONTROLLER

```
use App\Jobs\UpdateUserProfileJob;
UpdateUserProfileJob::dispatch($userId, $name, $email);
```

16) OBSERVER

1. php artisan make:observer UserObserver –model=User

2. CHOOSE FUNCTION TO USE IN THAT OBSERVER CLASS

1. creating - called before a new record is saved
2. created - called after a new record has been saved
3. updating - called before an existing record is updated
4. updated - called after an existing record has been updated
5. saving - called before a record is saved, both for new and existing records
6. saved - called after a record has been saved, both for new and existing records
7. deleting - called before an existing record is deleted
8. deleted - called after an existing record has been deleted
9. restoring - called before a soft-deleted record is restored
10. restored - called after a soft-deleted record has been restored

3. REGISTER THE OBSERVER in the **AppServiceProvider**

```
public function boot()
{
    User::observe(UserObserver::class);
}
```

17) EVENT & LISTENER

1. CREATE EVENT (just use to handle parameter - construct)

php artisan make:event UserRegistered

2. CREATE LISTENER (just use to execute the action - handle)

php artisan make:listener SendWelcomeEmail --event=UserRegistered

3. REGISTER THE EVENT AND LISTENER in **app/Providers/EventServiceProvider.php / Auto Discovery**

```
protected $listen = [

        UserRegistered::class => [
            SendWelcomeEmail::class,
        ],

    ];

@

public function shouldDiscoverEvent()
{
            return true;
}
```

4. CALL EVENT IN CONTROLLER

```
UserRegistered::dispatch( $parameter );
```

5. TO ADD ANYMORE LISTENER JUST DO STEP 2

18) MIDDLEWARE + REGISTER + LOGIN & LOGOUT

MIDDLEWARE

1) Create Middleware

php artisan make:middleware MyMiddleware

2) Register Middleware in kernel.php

```
protected $routeMiddleware = [
        'MyMiddleware' => \App\Http\Middleware\MiddlewareName::class
];
```

3) Middleware Class (App -> Http -> Middleware)

```
  public function handle($request, Closure $next)
  {
    if ( Auth :: user() -> role_id !=1){
        abort(404);
     }
    return $next($request);
  }
```

4) Use the middleware to a specific route

```
Route::get('/logout')->middleware(['auth','admin', MyMiddleware]);
@
Route::middleware(['MyMiddleware'])->group(function () {
  // routes
 // routes
});
```

5) use the middleware on specific controllers

```
  public function __construct()
  {       $this->middleware('mymiddleware');   }

  public function show(Request $request, $id)
  {       $this->middleware('mymiddleware');   }
```

REGISTER , LOGIN AND LOGOUT

A) REGISTER

1) In View

```
<form method="POST" action="/register">
    @csrf
    <label for="name">Name:</label>
    <input type="text" id="name" name="name" required>
    <label for="email">Email:</label>
    <input type="email" id="email" name="email" required>

    <label for="password">Password:</label>
    <input type="password" id="password" name="password" required>
    <button type="submit">Register</button>
</form>
```

2) In Controller

```
class RegisterController extends Controller
{
    public function register(Request $request)
    {
        $validatedData = $request->validate([
            'name' => 'required|string|max:255',
            'email' => 'required|string|email|max:255|unique:users',
            'password' => 'required|string|min:6|confirmed',
        ]);

        $user = User::create([
            'name' => $validatedData['name'],
            'email' => $validatedData['email'],
            'password' => Hash::make($validatedData['password']),
        ]);

        auth()->login($user);
        return redirect('/dashboard');
    }
```

```
}
```

3. Route

```
Route::post('/register', 'RegisterController@register');
```

B) LOGIN & LOGOUT

1. Login View

```
@if (Auth::check())
   <p>Welcome, {{ Auth::user()->name }}</p>
   <a href="/logout">Logout</a>
@else
   <form method="POST" action="/login">
      @csrf
      <label for="email">Email</label>
      <input type="email" name="email" id="email" required>

      <label for="password">Password</label>
      <input type="password" name="password" id="password" required>

      <button type="submit">Login</button>
   </form>
@endif
```

2. In Route

```
Route::post('/login', 'LoginController@login');
```

3. Login Controller

```
class LoginController extends Controller
{
   public function login(Request $request)
   {
      $credentials = $request->only('email', 'password');          // Get the email and password from the request

      // Attempt to log the user in with the given credentials
      if (Auth::attempt($credentials)) {
```

```
            return redirect()->intended('dashboard');            // If the login is successful, redirect  to  intended page
        }

        // If the login is not successful, redirect the user back to the login page with the email field filled in
        return redirect()->back()->withInput($request->only('email'));
    }
}
```

4. In Route

```
Route::post('/logout, 'LogoutController@logout);
```

5. LogOut Controller

```
class LogoutController extends Controller
{
    public function logout()
    {
        // Log the user out
        Auth::logout();
        return redirect('/');
    }
}
```

9) OTHERS

SOFT DELETE

1) Install the "laravel-soft-deletes" package via composer:

composer require jenssegers/laravel-soft-deletes

2) Add the "SoftDeletes" trait to the models you want to use soft delete on

```
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\SoftDeletes;

class YourModel extends Model {
    use SoftDeletes;
}
```

3) Migration - Add a "deleted_at" column to the corresponding table in your database

php artisan make:migration add_soft_delete_to_column_to_Students_Table

```
class AddSoftDeletesToYourTable extends Migration
{
    public function up()
    {
        Schema::table('your_table', function (Blueprint $table) {
            $table->softDeletes();
        });
    }

    public function down()
    {
        Schema::table('your_table', function (Blueprint $table) {
            $table->dropColumn('deleted_at');
        });
    }
}
```

4) Migrate the files

php artisan migrate

5) To retrieve only the models that have not been "soft deleted" and To retrieve only the models that have been "soft deleted",

```
$models = YourModel::withTrashed()->get();

$models = YourModel::onlyTrashed()->get();
```

```
public function index()

{
        $student = Student::withTrashed()->get();
        @
        $student = Student::onlyTrashed()->get();

        return view('student' , ['student' => $student]);

}
```

6) Restore Function

```
public function restore ($id)
{
        $student = Student :: withTrashed()->where('id' , $id)->restore();

        return redirect('someURL');
}
```

SLUG (to create a pretty and safer url)

1. Create migration file

php artisan make:migration create add_slug_to_students _table

2. In Migration File

```
public function up()
   {
      Schema::create('flights', function (Blueprint $table) {
            $table->string('slug' , 255) -> nullable() ->after('name');
      });
   }

public function down()
{
     Schema::table('flights', function (Blueprint $table) {
            if ( Schema :: hasColumn ( ' students ' , ' slug ' ) ){
                        $table->dropColumn('slug');
      });

}
```

3) Migrate the file

php artisan migrate

4) In controller

```
$request['slug'] = str :: slug ( $request->name , '-' ) ;
```

20) UNIT TEST

First : Create a test file

- php artisan make:test WhateverTest –unit

Second : Create a function

```
public function test_Update()
{
        //anycode here
}
```

Third : Create data for a model

```
$itemData = [
        'name' => 'item1',
        'description' => 'This is item1',
        'price' => 10,
 ];
```

Fourth : Check the route (get / post) + provide any data that the routes need

```
$updateData = [
        'name' => 'item2',
        'description' => 'This is item2',
        'price' => 20,
];

$response = $this->put(route('item.update', $item), $updateData);    @
$response = $this->get(route('item.view', $item));
```

Fifth : debug and test the code using “Assert”

1. RETURN REDIRECT

```
// Assert that the user is redirected to the item list view
$response->assertRedirect(route('item.list'));
$response->assertStatus(302);
```

2. RETURN VIEW

```
// Assert that the response has a HTTP status code of 200 + NOT FOR REDIRECT JUST FOR RETURN VIEW(‘any URL’)
$this->assertEquals(200, $response->status());

// Assert that the response has the correct view name
$this->assertViewIs('items.view');

// Assert that the view has the item data
$this->assertViewHas('item', $item);
```

3. CHECK DATABASE

```
// Assert that the item's data was added to the database
$this->assertDatabaseHas('items', $itemData);

// Assert that the item's data was deleted from the database
$this->assertDatabaseMissing('items', [
    'id' => $item->id
]);
```

4. CHECK SESSION

```
// Assert that the session has a success message
$this->assertSessionHas('success', 'Item added successfully');
```

1. Index/View

```
public function testIndex()
{
    $user = User::where('email', 'resitkusolution@gmail.com')->first();
    $this->actingAs($user);

    $response = $this->get(route('force-update.index'));
    $response->assertStatus(200);
    $response->assertViewIs('pages.force-update.index');
}
```

2. Store

```
public function testStore()
{
    $admin = User::where('email', 'resitkusolution@gmail.com')->first();
    $this->actingAs($admin);

    $itemData = [
            'status' => 1,
           'version' => '9.99',
     ];

      $this->post(route('force-update.store'), $itemData);
        ->assertRedirect(route('force-update.index'))
        ->assertSessionHasNoErrors();

        $this->assertDatabaseHas('force_updates', $itemData);
}
```

3. Update

```
public function testUpdateProfile()
  {
     $admin = User::where('email', 'resitkusolution@gmail.com')->first();
     $this->actingAs($admin);

     $consumer = new ConsumerDetail;
     $consumer->address = 'Initial Address';
     $consumer->phone = 'Initial Phone';
     $consumer->type_of_business = 'Initial Business Type';
     $consumer->affiliate_id = 'Initial Affiliate ID';
     $consumer->save();

    $request = new Request([
                    'address' => '123 Main St',
                    'phone' => '555-555-5555',
                    'type_of_business' => 'Retail',
                    'affiliate_id' => '123456789'
     ]);

     $response = $this->post('/user-profile-update/' . $consumer->id, $request->all());
     $response->assertStatus(302);

  }
```

4. Delete

```
public function testDestroy()
{
    $admin = User::where('email', 'resitkusolution@gmail.com')->first();
    $this->actingAs($admin);

    $item = Item::create([
        'name' => 'item1',
        'description' => 'This is item1',
        'price' => 10,
    ]);

    $response = $this-->delete(route('force-update.destroy', $item));

    $response->assertRedirect(route('force-update.index'));
    $this->assertDatabaseMissing('force_updates', [
        'id' => $forceUpdate->id,
    ]);
}
```

*force_updates = name of table in db

5. TEST A MODEL

```
    public function testCreateUser()
    {
        // Create a new user instance
        $user = new User();

        // Set the user's properties
        $user->name = 'John Doe';
        $user->email = 'johndoe@example.com';
        $user->password = bcrypt('password');

        // Save the user to the database
        $user->save();

        // Assert that the user was saved to the database
        $this->assertDatabaseHas('users', [
            'name' => 'John Doe',
            'email' => 'johndoe@example.com'
        ]);
    }
}
```

LARAVEL USEFUL PACKAGE

1. LARAVEL EXCEL
2. LARAVEL BREEZE (LOGIN,LOGOUT)
3. LARAVEL SCOUT (SEARCH)
4. LARAVEL PINT (CODE PRETTY)
5. LARAVEL DEBUGBAR / LARAVEL SCOUT (MONITOR TIME QUERY)
6. LARAVEL SOCIALIZE (LOGIN WITH GITHUB)