**////////////////////////////////Start/////////////////////////////\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

# Database: Migrations

* [**Introduction**](https://laravel.com/docs/5.3/migrations#introduction)
* [**Generating Migrations**](https://laravel.com/docs/5.3/migrations#generating-migrations)
* [**Migration Structure**](https://laravel.com/docs/5.3/migrations#migration-structure)
* [**Running Migrations**](https://laravel.com/docs/5.3/migrations#running-migrations)
  + [Rolling Back Migrations](https://laravel.com/docs/5.3/migrations#rolling-back-migrations)
* [**Tables**](https://laravel.com/docs/5.3/migrations#tables)
  + [Creating Tables](https://laravel.com/docs/5.3/migrations#creating-tables)
  + [Renaming / Dropping Tables](https://laravel.com/docs/5.3/migrations#renaming-and-dropping-tables)
* [**Columns**](https://laravel.com/docs/5.3/migrations#columns)
  + [Creating Columns](https://laravel.com/docs/5.3/migrations#creating-columns)
  + [Column Modifiers](https://laravel.com/docs/5.3/migrations#column-modifiers)
  + [Modifying Columns](https://laravel.com/docs/5.3/migrations#modifying-columns)
  + [Dropping Columns](https://laravel.com/docs/5.3/migrations#dropping-columns)
* [**Indexes**](https://laravel.com/docs/5.3/migrations#indexes)
  + [Creating Indexes](https://laravel.com/docs/5.3/migrations#creating-indexes)
  + [Dropping Indexes](https://laravel.com/docs/5.3/migrations#dropping-indexes)
  + [Foreign Key Constraints](https://laravel.com/docs/5.3/migrations#foreign-key-constraints)

## [Introduction](https://laravel.com/docs/5.3/migrations" \l "introduction)

Migrations are like version control for your database, allowing your team to easily modify and share the application's database schema. Migrations are typically paired with Laravel's schema builder to easily build your application's database schema. If you have ever had to tell a teammate to manually add a column to their local database schema, you've faced the problem that database migrations solve.

The Laravel Schema [facade](https://laravel.com/docs/5.3/facades) provides database agnostic support for creating and manipulating tables across all of Laravel's supported database systems.

## [Generating Migrations](https://laravel.com/docs/5.3/migrations" \l "generating-migrations)

To create a migration, use the make:migration [Artisan command](https://laravel.com/docs/5.3/artisan):

php artisan make:migration create\_users\_table

The new migration will be placed in your database/migrations directory. Each migration file name contains a timestamp which allows Laravel to determine the order of the migrations.

The --table and --create options may also be used to indicate the name of the table and whether the migration will be creating a new table. These options simply pre-fill the generated migration stub file with the specified table:

php artisan make:migration create\_users\_table --create=users

php artisan make:migration add\_votes\_to\_users\_table --table=users

If you would like to specify a custom output path for the generated migration, you may use the --path option when executing the make:migration command. The given path should be relative to your application's base path.

## [Migration Structure](https://laravel.com/docs/5.3/migrations" \l "migration-structure)

A migration class contains two methods: up and down. The up method is used to add new tables, columns, or indexes to your database, while the down method should simply reverse the operations performed by the up method.

Within both of these methods you may use the Laravel schema builder to expressively create and modify tables. To learn about all of the methods available on the Schema builder, [check out its documentation](https://laravel.com/docs/5.3/migrations#creating-tables). For example, this migration example creates a flights table:

<?php

use Illuminate\Support\Facades\Schema;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Database\Migrations\Migration;

class CreateFlightsTable extends Migration

{

/\*\*

\* Run the migrations.

\*

\* @return void

\*/

public function up()

{

Schema::create('flights', function (Blueprint $table) {

$table->increments('id');

$table->string('name');

$table->string('airline');

$table->timestamps();

});

}

/\*\*

\* Reverse the migrations.

\*

\* @return void

\*/

public function down()

{

Schema::drop('flights');

}

}

## [Running Migrations](https://laravel.com/docs/5.3/migrations" \l "running-migrations)

To run all of your outstanding migrations, execute the migrate Artisan command:

php artisan migrate

If you are using the [Homestead virtual machine](https://laravel.com/docs/5.3/homestead), you should run this command from within your virtual machine.

#### Forcing Migrations To Run In Production

Some migration operations are destructive, which means they may cause you to lose data. In order to protect you from running these commands against your production database, you will be prompted for confirmation before the commands are executed. To force the commands to run without a prompt, use the --force flag:

php artisan migrate --force

### Rolling Back Migrations

To rollback the latest migration operation, you may use the rollback command. This command rolls back the last "batch" of migrations, which may include multiple migration files:

php artisan migrate:rollback

You may rollback a limited number of migrations by providing the step option to the rollbackcommand. For example, the following command will rollback the last five migrations:

php artisan migrate:rollback --step=5

The migrate:reset command will roll back all of your application's migrations:

php artisan migrate:reset

#### Rollback & Migrate In Single Command

The migrate:refresh command will roll back all of your migrations and then execute the migratecommand. This command effectively re-creates your entire database:

php artisan migrate:refresh

// Refresh the database and run all database seeds...

php artisan migrate:refresh --seed

You may rollback & re-migrate a limited number of migrations by providing the step option to the refresh command. For example, the following command will rollback & re-migrate the last five migrations:

php artisan migrate:refresh --step=5

## [Tables](https://laravel.com/docs/5.3/migrations" \l "tables)

### Creating Tables

To create a new database table, use the create method on the Schema facade. The create method accepts two arguments. The first is the name of the table, while the second is a Closure which receives a Blueprint object that may be used to define the new table:

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

$table->increments('id');

});

Of course, when creating the table, you may use any of the schema builder's [column methods](https://laravel.com/docs/5.3/migrations#creating-columns) to define the table's columns.

#### Checking For Table / Column Existence

You may easily check for the existence of a table or column using the hasTable and hasColumnmethods:

if (Schema::hasTable('users')) {

//

}

if (Schema::hasColumn('users', 'email')) {

//

}

#### Connection & Storage Engine

If you want to perform a schema operation on a database connection that is not your default connection, use the connection method:

Schema::connection('foo')->create('users', function ($table) {

$table->increments('id');

});

You may use the engine property on the schema builder to define the table's storage engine:

Schema::create('users', function ($table) {

$table->engine = 'InnoDB';

$table->increments('id');

});

### Renaming / Dropping Tables

To rename an existing database table, use the rename method:

Schema::rename($from, $to);

To drop an existing table, you may use the drop or dropIfExists methods:

Schema::drop('users');

Schema::dropIfExists('users');

#### Renaming Tables With Foreign Keys

Before renaming a table, you should verify that any foreign key constraints on the table have an explicit name in your migration files instead of letting Laravel assign a convention based name. Otherwise, the foreign key constraint name will refer to the old table name.

## [Columns](https://laravel.com/docs/5.3/migrations" \l "columns)

### Creating Columns

The table method on the Schema facade may be used to update existing tables. Like the createmethod, the table method accepts two arguments: the name of the table and a Closure that receives a Blueprint instance you may use to add columns to the table:

Schema::table('users', function ($table) {

$table->string('email');

});

#### Available Column Types

Of course, the schema builder contains a variety of column types that you may specify when building your tables:

| **Command** | **Description** |
| --- | --- |
| $table->bigIncrements('id'); | Incrementing ID (primary key) using a "UNSIGNED BIG INTEGER" equivalent. |
| $table->bigInteger('votes'); | BIGINT equivalent for the database. |
| $table->binary('data'); | BLOB equivalent for the database. |
| $table->boolean('confirmed'); | BOOLEAN equivalent for the database. |
| $table->char('name', 4); | CHAR equivalent with a length. |
| $table->date('created\_at'); | DATE equivalent for the database. |
| $table->dateTime('created\_at'); | DATETIME equivalent for the database. |
| $table->dateTimeTz('created\_at'); | DATETIME (with timezone) equivalent for the database. |
| $table->decimal('amount', 5, 2); | DECIMAL equivalent with a precision and scale. |
| $table->double('column', 15, 8); | DOUBLE equivalent with precision, 15 digits in total and 8 after the decimal point. |
| $table->enum('choices', ['foo', 'bar']); | ENUM equivalent for the database. |
| $table->float('amount', 8, 2); | FLOAT equivalent for the database, 8 digits in total and 2 after the decimal point. |
| $table->increments('id'); | Incrementing ID (primary key) using a "UNSIGNED INTEGER" equivalent. |
| $table->integer('votes'); | INTEGER equivalent for the database. |
| $table->ipAddress('visitor'); | IP address equivalent for the database. |
| $table->json('options'); | JSON equivalent for the database. |
| $table->jsonb('options'); | JSONB equivalent for the database. |
| $table->longText('description'); | LONGTEXT equivalent for the database. |
| $table->macAddress('device'); | MAC address equivalent for the database. |
| $table->mediumIncrements('id'); | Incrementing ID (primary key) using a "UNSIGNED MEDIUM INTEGER" equivalent. |
| $table->mediumInteger('numbers'); | MEDIUMINT equivalent for the database. |
| $table->mediumText('description'); | MEDIUMTEXT equivalent for the database. |
| $table->morphs('taggable'); | Adds unsigned INTEGER taggable\_id and STRING taggable\_type. |
| $table->nullableTimestamps(); | Same as timestamps(). |
| $table->rememberToken(); | Adds remember\_token as VARCHAR(100) NULL. |
| $table->smallIncrements('id'); | Incrementing ID (primary key) using a "UNSIGNED SMALL INTEGER" equivalent. |
| $table->smallInteger('votes'); | SMALLINT equivalent for the database. |
| $table->softDeletes(); | Adds nullable deleted\_at column for soft deletes. |
| $table->string('email'); | VARCHAR equivalent column. |
| $table->string('name', 100); | VARCHAR equivalent with a length. |
| $table->text('description'); | TEXT equivalent for the database. |
| $table->time('sunrise'); | TIME equivalent for the database. |
| $table->timeTz('sunrise'); | TIME (with timezone) equivalent for the database. |
| $table->tinyInteger('numbers'); | TINYINT equivalent for the database. |
| $table->timestamp('added\_on'); | TIMESTAMP equivalent for the database. |
| $table->timestampTz('added\_on'); | TIMESTAMP (with timezone) equivalent for the database. |
| $table->timestamps(); | Adds nullable created\_at and updated\_at columns. |
| $table->timestampsTz(); | Adds nullable created\_at and updated\_at (with timezone) columns. |
| $table->unsignedBigInteger('votes'); | Unsigned BIGINT equivalent for the database. |
| $table->unsignedInteger('votes'); | Unsigned INT equivalent for the database. |
| $table->unsignedMediumInteger('votes'); | Unsigned MEDIUMINT equivalent for the database. |
| $table->unsignedSmallInteger('votes'); | Unsigned SMALLINT equivalent for the database. |
| $table->unsignedTinyInteger('votes'); | Unsigned TINYINT equivalent for the database. |
| $table->uuid('id'); | UUID equivalent for the database. |

### Column Modifiers

In addition to the column types listed above, there are several column "modifiers" you may use while adding a column to a database table. For example, to make the column "nullable", you may use the nullable method:

Schema::table('users', function ($table) {

$table->string('email')->nullable();

});

Below is a list of all the available column modifiers. This list does not include the [index modifiers](https://laravel.com/docs/5.3/migrations#creating-indexes):

| **Modifier** | **Description** |
| --- | --- |
| ->after('column') | Place the column "after" another column (MySQL Only) |
| ->comment('my comment') | Add a comment to a column |
| ->default($value) | Specify a "default" value for the column |
| ->first() | Place the column "first" in the table (MySQL Only) |
| ->nullable() | Allow NULL values to be inserted into the column |
| ->storedAs($expression) | Create a stored generated column (MySQL Only) |
| ->unsigned() | Set integer columns to UNSIGNED |
| ->virtualAs($expression) | Create a virtual generated column (MySQL Only) |

### Modifying Columns

#### Prerequisites

Before modifying a column, be sure to add the doctrine/dbal dependency to your composer.jsonfile. The Doctrine DBAL library is used to determine the current state of the column and create the SQL queries needed to make the specified adjustments to the column:

composer require doctrine/dbal

#### Updating Column Attributes

The change method allows you to modify some existing column types to a new type or modify the column's attributes. For example, you may wish to increase the size of a string column. To see the change method in action, let's increase the size of the name column from 25 to 50:

Schema::table('users', function ($table) {

$table->string('name', 50)->change();

});

We could also modify a column to be nullable:

Schema::table('users', function ($table) {

$table->string('name', 50)->nullable()->change();

});

The following column types can not be "changed": char, double, enum, mediumInteger, timestamp, tinyInteger, ipAddress, json, jsonb, macAddress, mediumIncrements, morphs, nullableTimestamps, softDeletes, timeTz, timestampTz, timestamps, timestampsTz, unsignedMediumInteger, unsignedTinyInteger, uuid.

#### Renaming Columns

To rename a column, you may use the renameColumn method on the Schema builder. Before renaming a column, be sure to add the doctrine/dbal dependency to your composer.json file:

Schema::table('users', function ($table) {

$table->renameColumn('from', 'to');

});

Renaming any column in a table that also has a column of type enum is not currently supported.

### Dropping Columns

To drop a column, use the dropColumn method on the Schema builder. Before dropping columns from a SQLite database, you will need to add the doctrine/dbal dependency to your composer.jsonfile and run the composer update command in your terminal to install the library:

Schema::table('users', function ($table) {

$table->dropColumn('votes');

});

You may drop multiple columns from a table by passing an array of column names to the dropColumn method:

Schema::table('users', function ($table) {

$table->dropColumn(['votes', 'avatar', 'location']);

});

Dropping or modifying multiple columns within a single migration while using a SQLite database is not supported.

## [Indexes](https://laravel.com/docs/5.3/migrations" \l "indexes)

### Creating Indexes

The schema builder supports several types of indexes. First, let's look at an example that specifies a column's values should be unique. To create the index, we can simply chain the unique method onto the column definition:

$table->string('email')->unique();

Alternatively, you may create the index after defining the column. For example:

$table->unique('email');

You may even pass an array of columns to an index method to create a compound index:

$table->index(['account\_id', 'created\_at']);

Laravel will automatically generate a reasonable index name, but you may pass a second argument to the method to specify the name yourself:

$table->index('email', 'my\_index\_name');

#### Available Index Types

| **Command** | **Description** |
| --- | --- |
| $table->primary('id'); | Add a primary key. |
| $table->primary(['first', 'last']); | Add composite keys. |
| $table->unique('email'); | Add a unique index. |
| $table->unique('state', 'my\_index\_name'); | Add a custom index name. |
| $table->unique(['first', 'last']); | Add a composite unique index. |
| $table->index('state'); | Add a basic index. |

### Dropping Indexes

To drop an index, you must specify the index's name. By default, Laravel automatically assigns a reasonable name to the indexes. Simply concatenate the table name, the name of the indexed column, and the index type. Here are some examples:

| **Command** | **Description** |
| --- | --- |
| $table->dropPrimary('users\_id\_primary'); | Drop a primary key from the "users" table. |
| $table->dropUnique('users\_email\_unique'); | Drop a unique index from the "users" table. |
| $table->dropIndex('geo\_state\_index'); | Drop a basic index from the "geo" table. |

If you pass an array of columns into a method that drops indexes, the conventional index name will be generated based on the table name, columns and key type:

Schema::table('geo', function ($table) {

$table->dropIndex(['state']); // Drops index 'geo\_state\_index'

});

### Foreign Key Constraints

Laravel also provides support for creating foreign key constraints, which are used to force referential integrity at the database level. For example, let's define a user\_id column on the posts table that references the id column on a users table:

Schema::table('posts', function ($table) {

$table->integer('user\_id')->unsigned();

$table->foreign('user\_id')->references('id')->on('users');

});

You may also specify the desired action for the "on delete" and "on update" properties of the constraint:

$table->foreign('user\_id')

->references('id')->on('users')

->onDelete('cascade');

To drop a foreign key, you may use the dropForeign method. Foreign key constraints use the same naming convention as indexes. So, we will concatenate the table name and the columns in the constraint then suffix the name with "\_foreign":

$table->dropForeign('posts\_user\_id\_foreign');

Or, you may pass an array value which will automatically use the conventional constraint name when dropping:

$table->dropForeign(['user\_id']);

You may enable or disable foreign key constraints within your migrations by using the following methods:

Schema::enableForeignKeyConstraints();

Schema::disableForeignKeyConstraints();

* [Documentation](https://laravel.com/docs)

* [Laracasts](https://laracasts.com/)

* [News](https://laravel-news.com/)

* [Forge](https://forge.laravel.com/)

* [Conference](http://laracon.us/)

* [Ecosystem](https://laravel.com/docs/5.3/migrations)

LARAVEL IS A TRADEMARK OF TAYLOR OTWELL. COPYRIGHT © TAYLOR OTWELL.

[DESIGNED BY](http://jackmcdade.com/)

* And your done (you may have to restart your terminal). Just type in "composer"
* and you should see the usage options.
* Tip: If you're using windows powershell you might get the following warning;
* '■@' is not recognized as an internal or external command,
* If that's the case simply delete the composer.bat file, create a new file, copy in the following contents and save as composer.bat:

@ECHO OFF

php "%~dp0composer.phar" %\*

den restart the pc.

**Laravel install**

* download zip file from github <https://github.com/laravel/laravel> and unzip and put in on wamp>www>folderproject>shift+right click>open command window here>composer install

**create project (u can project ,in 2 ways one is described in 1 and other is described in 2)**

1. go to wamp>www>hold shift and click right mouse ->open command window here->paste composer create-project laravel/laravel in command prompt
   1. then go to laravel project then shift+ right click ,open command window here>write **php artisan serve**
2. download zip file from github <https://github.com/laravel/laravel> and unzip and put in on wamp>www>folderproject>shift+right click>open command window here>**composer install**

**Config Laravel**

* after install laravel then goes to wamp>www>laravelproject
* copy from here .env.example ,and paste it here then .env - Copy.example will create,
* then open sublime ,go to .env - Copy.example file right click and rename to .env and enter
* then goes to wamp>www>laravelproject ,shift+right click ,write php artisan key:generate

**To move Laravelproject folder one pc to another pc**

* to move laravelproject one pc to another ,first check wamp,composer are intalled ,
* then put laravelproject on **wamp/www/** directory ,then in project ,click shift+right click->open window here-> write **composer install**
* copy env.example ,and paste it here then .env - Copy.example will create,
* coz after write composer install .env will be removed
* set .env like before
* write **php artisan key:generate**

**laravel public folder Remove**

* <https://www.youtube.com/watch?v=zQsyxumCZqQ>
* Go to mainproject/public>>

1. .htacess
2. favicon.ico
3. index.php
4. robots.txt
5. web.config

1.      cut these 5 files from public folder,and then paste on the main project folder that’s means out side of public folder… mainproject/files

2.      Next after paste ,open index.php ,modify

        require \_\_DIR\_\_.'/…/bootstrap/autoload.php'; to

         require \_\_DIR\_\_.'/bootstrap/autoload.php';

 3. modify

$app = require\_once \_\_DIR\_\_.'/../bootstrap/app.php'; to

$app = require\_once \_\_DIR\_\_.'/bootstrap/app.php';

**import tables in db by migration command**

1. Php artisan make:model userdefined\_modelname

**Laravel class ‘input’ not found solution**

Add 'Input' => Illuminate\Support\Facades\Input::class, in config/app.php alias

**Laravel class ‘form’ not found solution**

<https://laravelcollective.com/docs/5.2/html>

1. Begin by installing this package through Composer. Edit your project's composer.json file to require laravelcollective/html.

"require": {

"laravelcollective/html": "5.2.\*"

}

1. Next, update Composer from the Terminal:

composer update

**[1 and 2 instead e composer require laravelcollective/html composer a paste korleo hobe]**

1. Next, add your new provider to the providers array of config/app.php:

'providers' => [

// ...

Collective\Html\HtmlServiceProvider::class,

// ...

],

1. Finally, add two class aliases to the aliases array of config/app.php:

'aliases' => [

// ...

'Form' => Collective\Html\FormFacade::class,

'Html' => Collective\Html\HtmlFacade::class,

// ...

],

* *Another way if package not install from composer*

<http://stackoverflow.com/questions/32666164/laravel-5-installing-form-and-html-classes-without-composer>

<https://github.com/LaravelCollective/html>

|  |  |
| --- | --- |
|  | **Installing laracollective/html:**  **1)** Download zip package from [extension's git repository](https://github.com/LaravelCollective/html)  **2)** Unpack contents and create this directory structure: laravelcollective/html/{contents of html-5.1 directory}  **3)** Copy this structure to the vendor folder in your Laravel installation.  **4)** In vendor/composer/autoload\_classmap.php add these lines  'Collective\\Html\\FormBuilder' => $vendorDir . '/laravelcollective/html/src/FormBuilder.php',  'Collective\\Html\\FormFacade' => $vendorDir . '/laravelcollective/html/src/FormFacade.php',  'Collective\\Html\\HtmlBuilder' => $vendorDir . '/laravelcollective/html/src/HtmlBuilder.php',  'Collective\\Html\\HtmlFacade' => $vendorDir . '/laravelcollective/html/src/HtmlFacade.php',  'Collective\\Html\\HtmlServiceProvider' => $vendorDir . '/laravelcollective/html/src/HtmlServiceProvider.php',  after  'ClassPreloader\\Parser\\NodeTraverser' => $vendorDir . '/classpreloader/classpreloader/src/Parser/NodeTraverser.php',  so that Laravel knows where to look for these classes when requested.  **5)** Add this line as a last element to return array(...) in vendor/composer/autoload\_files.php  'f18cc91337d49233e5754e93f3ed9ec3' => $vendorDir . '/laravelcollective/html/src/helpers.php',  **6)** Add this line to return array(...) in vendor/composer/autoload\_psr4.php  // 'Doctrine\\Instantiator\\' => array($vendorDir . '/doctrine/instantiator/src/Doctrine/Instantiator'),  'Collective\\Html\\' => array($vendorDir . '/laravelcollective/html/src'),  // 'ClassPreloader\\' => array($vendorDir . '/classpreloader/classpreloader/src'),  **7)** Add providers to providers array of config/app.php:  'providers' => [  // ...  Collective\Html\HtmlServiceProvider::class,  // ...  ],  **8)** Add two class aliases to the aliases array of config/app.php:  'aliases' => [  // ...  'Form' => Collective\Html\FormFacade::class,  'Html' => Collective\Html\HtmlFacade::class,  // ...  ],  **Laravel Controller**   * How to create controller   Php artisan make:controller userdefined\_controller\_name   * How to access view ,suppose In AdminController.php   Public function login(){  Return view(‘admin.login’);  }  Then Route.php  Route::get(‘/login’,array(‘as’=>’login’,’uses’=>’AdminController@login’);  Route::get(‘/url,array(‘as’=>’login’,’uses’=>’controllername@functionname);   * To **retrieve all data from database** use modelname::all() and use App\Customer; in controller upper part   public function index()  {  $customers=Customer::all();  return view('customer',['customers'=>$customers]);  }  -**various ways of retrieve data’s**  1. $customers=Customer::all();  2. $product\_lists=Product::lists('productname','id');  3. $data=Product::select('price')->where('id',$request->id)->first();  4. $students=DB::table('tblstudent')->get();  5. $data = Project::all(array("id", "project\_title"));  6. $data\_qry= DB::select("SELECT ex.\*,  proj.project\_title,  a.first\_name as 'addedBy',  p.first\_name as 'paidTo'  FROM expenses ex, projects proj, users p, users a  WHERE proj.id= ex.project\_id AND  ex.paid\_to= p.id AND  ex.added\_by= a.id AND  ex.expenses\_date BETWEEN ? AND ?  AND project\_id=?",[$var\_start\_date,$var\_end\_date,$var\_project\_name\_invoice]);  7. $data=DB::table('ch\_info')->select('name', 'ch\_id as ch','addr','id')->take(100)->get();  8.$datapr=DB::select("SELECT pr.product\_id as prid,  pr.p\_code as pcode,  pr.brand\_name as brand,  pr.pack\_s as ppack,  pr.p\_group as pgroup,  pr\_price.T\_P as price  FROM product\_info pr, product\_info\_price pr\_price WHERE pr.p\_code=pr\_price.p\_code");   * To redirect same page use return back(); in controller   Or  return redirect()->back()->with('success','Expenses Created Successfully');   * To insert data model\_name::insert($data);   or  $customers=new Incustomer;  $customers->firstname=$request->fn;  $customers->lastname=$request->ln;  $customers->save();  Or  $response=Salary::create($data);   * To return view with data one is compact another []   $product\_lists=Product::lists('productname','id');  return view('invoice.info',compact('product\_lists'));  or  $customers=Customer::all();  return view('customer',['customers'=>$customers]);  or  $data = Project::all(array("id", "project\_title"));  $users = EmployeeModel::all(array("id", "first\_name"));  return view('addsalary')->withProjects($data)->withUsers($users);  or  return view('projectBudget' )->with('success',"Budget Added")->with('project\_title', $response->project\_title)->with('project\_id', $response->id)->with('budget',2000);  or  $usersdata=DB::table('users')->get();  $projectdata =DB::table('projects')->get();  return view('view-invoice',compact('usersdata','projectdata'));   * To update data in database table   DB::update('update users set votes = 100 where name = ?', ['John']);  Or  DB::table('users')->where('id', 1)->update(['price' => 1]);  Composer laravel cmd:   * Php artisan make:auth (Authentication) * Php artisan migrate // Run migration * Php artisan make:controller userdefined\_controller\_name * Php artisan key:generate * Composer install // For install composer packages * php artisan make:request **StoreBlogPostRequest** * Php artisan make:model userdefined\_model\_name   **Prob :Cannot access empty property**  Solution:  foreach ($object->someArray as $key->$val) {  // do something  }  Of course, I should have had:  foreach ($object->someArray as $key=>$val) {  // do something  } TokenMismatchException in VerifyCsrfToken.php line 67 in Laravel 5.2 Add <input type="hidden" name="\_token" value="{{ csrf\_token() }}"> before every input field like  <input type="hidden" name="\_token" value="{{ csrf\_token() }}">  <input type="text" name="last\_name" id="last\_name" placeholder="Last Name" class="form-control">  **How to use Datatable In Laravel 5.1**   * Database: datatable * table-tblstudent   -id  -first\_name  -last\_name  -pob  -phone   * composer require yajra/laravel-datatables-oracle * In provider (config/app.php)   Yajra\Datatables\DatatablesServiceProvider::class,   * In aliases(config/app.php)   'Datatables'=>Yajra\Datatables\Datatables::class,   * php artisan vendor:publish   -------------------------downloading coding below---------------------------------  -cdn.datatables.net/1.10.10/css/jquery.dataTables.css ---and put this file inside of assets/css/  -cdn.datatables.net/1.10.10/js/jquery.dataTables.js ---and put this file inside of assets/js/  -https://jquery.com/download/ ---and put this file inside of assets/js/  -https://github.com/DataTables/DataTables/tree/master/media/images ---and put this file inside of assets/ images/  favicon.ico sort\_asc.png sort\_asc\_disabled.png sort\_both.png sort\_desc.png sort\_desc\_disabled.png   * add following in before of body end tag   {!!Html::script('../resources/assets/js/jquery-1.12.1.js')!!}  {!!Html::script('../resources/assets/js/jquery.dataTables.js')!!}  <script type="text/javascript">  $('.student').dataTable({    });  </script>  </body>  </html>  And add following codes inside of header part  {!!Html::style('../resources/assets/css/jquery.dataTables.css')!!}  {!!Html::style('../resources/assets/css/bootstrap.css')!!}   * php artisan make:controller StudentController * use use DB; in controller's first * add this function in controller   public function index(){  $students=DB::table('tblstudent')->get();  return view('student',compact('student'));  }   * create student.blade.php in resources/views * in route.php ,write Route::get('/datTable',array('as'=>'viewStudent','uses'=>'StudentController@index'));  Setting Sublime Text up For Laravel Blade <https://www.youtube.com/watch?v=FeuZtBdS20c>   * Go to below link   <https://packagecontrol.io/installation#st3>   * copy following lines for sublime text3   import urllib.request,os,hashlib; h = '2915d1851351e5ee549c20394736b442' + '8bc59f460fa1548d1514676163dafc88'; pf = 'Package Control.sublime-package'; ipp = sublime.installed\_packages\_path(); urllib.request.install\_opener( urllib.request.build\_opener( urllib.request.ProxyHandler()) ); by = urllib.request.urlopen( 'http://packagecontrol.io/' + pf.replace(' ', '%20')).read(); dh = hashlib.sha256(by).hexdigest(); print('Error validating download (got %s instead of %s), please try manual install' % (dh, h)) if dh != h else open(os.path.join( ipp, pf), 'wb' ).write(by)   * Open sublime ,View > Show Console menu and paste upper lines into here * Then close sublime * The again open sublime and go to Tools>common palette (ctlr+shift+p) * Write package install ,click it,write laravel hightight and click it  Class 'App\Http\Controllers\Validator' not foundGenerating migration php artisan make:migration create\_users\_table  php artisan migrate |
|  |  |