Basic CRUD operation using Laravel

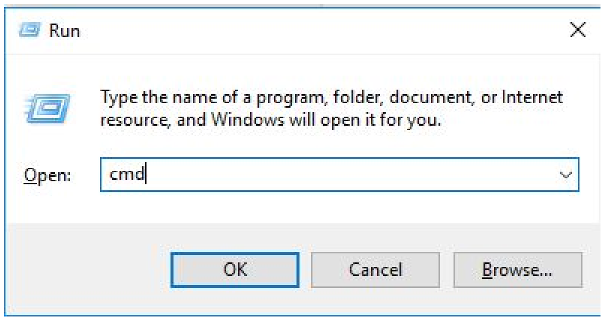
The goal of this tutorial is, we will create a crud operation for tasks. That means there will be table tasks in our database. We will collect data from user using a form submitted by user, we will insert data into database, we will view data in a tabular form, user also can update and delete the data. So let's start.

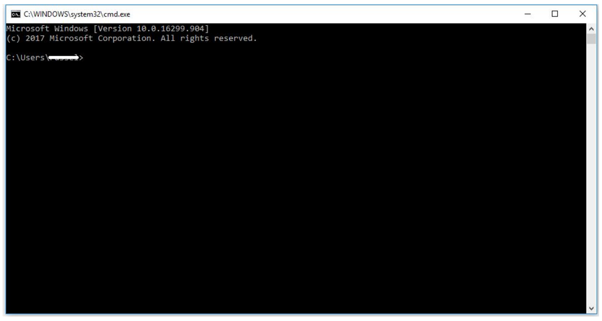
Requirements

1. Local server with php >=7.1 (for me. Xampp with PHP Version 7.3.0)
2. Composer
3. Laravel version 5.7
4. IDE (I am using PhpStorm)

I assume you have installed xampp/wamp with the required php version. Also you have installed Composer.

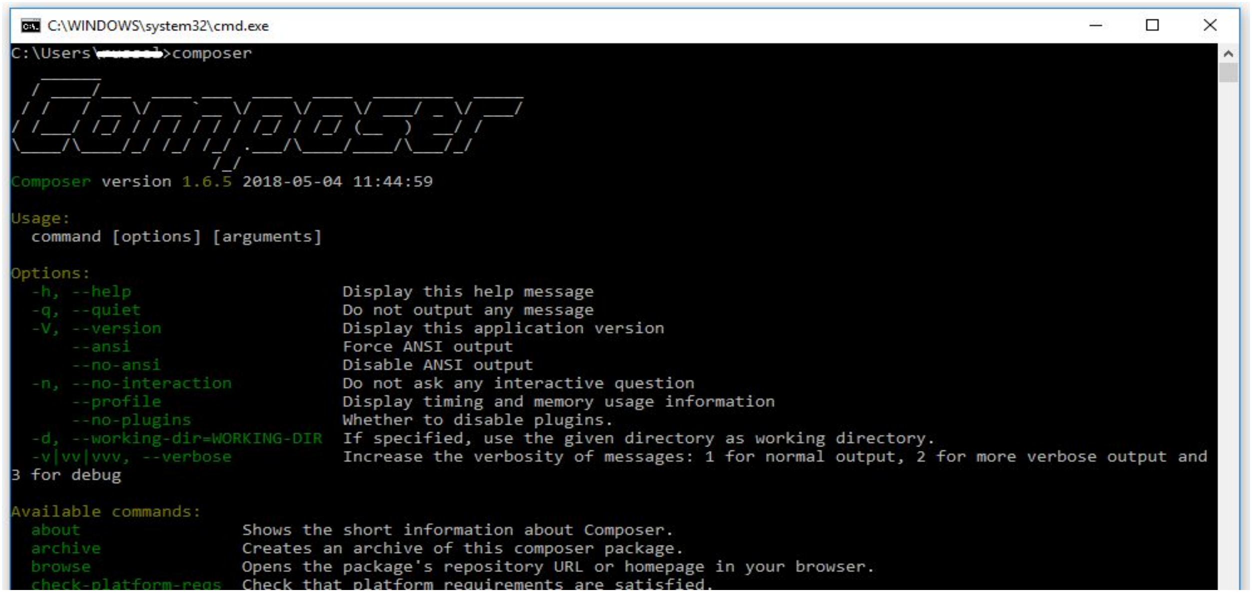
Step 1: Let's check the composer. Open command window (Press window+r, Itype ***cmd***, then press enter) .


Run
Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.
Open:
cmd
OK
Cancel
Browse...



```
Microsoft Windows [Version 10.0.16299.904]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\[redacted]>
```

Now write ***composer*** and press enter. You will see like the following figure:



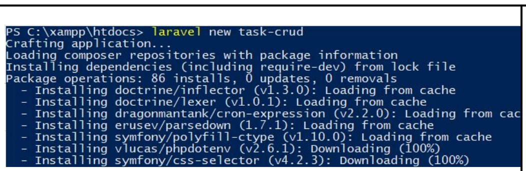
```
C:\Users\[redacted]>composer
   ______
  / ____/___  ____ ___  ____  ____  ________  _____
 / /   / __ \/ __ `__ \/ __ \/ __ \/ ___/ _ \/ ___/
/ /___/ /_/ / / / / / / /_/ / /_/ (__  )  __/ /
\____/\____/_/ /_/ /_/ .___/\____/____/\___/_/
                    /_/
Composer version 1.6.5 2018-05-04 11:44:59

Usage:
  command [options] [arguments]

Options:
  -h, --help                     Display this help message
  -q, --quiet                    Do not output any message
  -V, --version                  Display this application version
      --ansi                     Force ANSI output
      --no-ansi                  Disable ANSI output
  -n, --no-interaction           Do not ask any interactive question
      --profile                  Display timing and memory usage information
      --no-plugins               Whether to disable plugins.
  -d, --working-dir=WORKING-DIR  If specified, use the given directory as working directory.
  -v|vv|vvv, --verbose           Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and
3 for debug

Available commands:
  about                Shows the short information about Composer.
  archive              Creates an archive of this composer package.
  browse               Opens the package's repository URL or homepage in your browser.
  check-platform-reqs  Check that platform requirements are satisfied.
```

If you see something like the above picture, that means you have composer. If you don't have composer the go to this link (https://getcomposer.org/download/) and install composer.

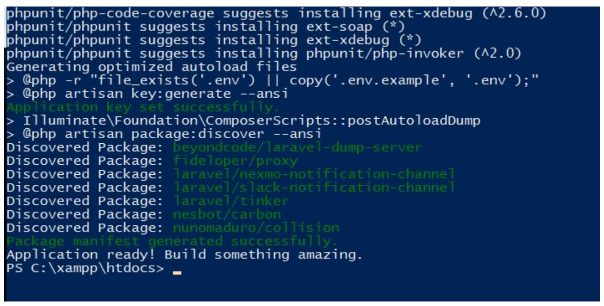
Step 2: Laravel Installation Open command window and write the following command

```
composer global require laravel/installer
```

Step 3: Project Creation Open command window and change directory to htdocs location. For me ***C:/xampp/htdocs*** and write the following command `laravel  new task-crud` and press enter. Wait until finish the project creation. You will see something like following

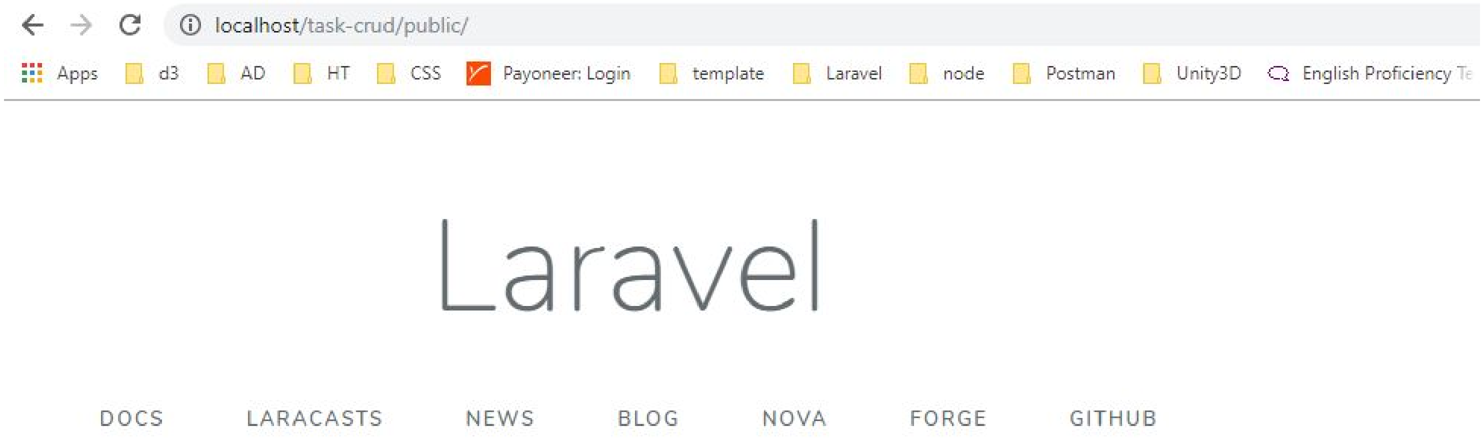


```
PS C:\xampp\htdocs> laravel new task-crud
Crafting application...
Loading composer repositories with package information
Installing dependencies (including require-dev) from lock file
Package operations: 86 installs, 0 updates, 0 removals
  - Installing doctrine/inflector (v1.3.0): Loading from cache
  - Installing doctrine/lexer (v1.0.1): Loading from cache
  - Installing dragonmantank/cron-expression (v2.2.0): Loading from cac
  - Installing erusev/parsedown (1.7.1): Loading from cache
  - Installing symfony/polyfill-ctype (v1.10.0): Loading from cache
  - Installing vlucas/phpdotenv (v2.6.1): Downloading (100%)
  - Installing symfony/css-selector (v4.2.3): Downloading (100%)
```



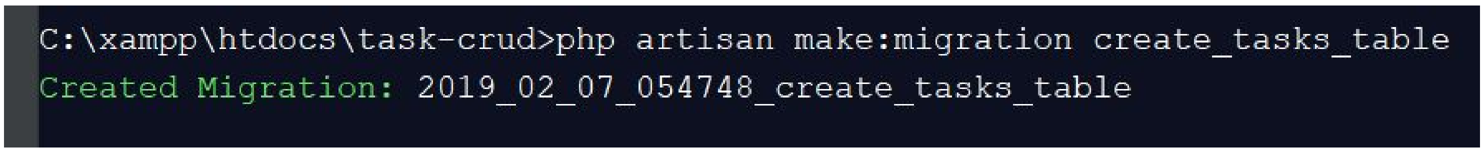
```
phpunit/php-code-coverage suggests installing ext-xdebug (^2.6.0)
phpunit/phpunit suggests installing ext-soap (*)
phpunit/phpunit suggests installing ext-xdebug (*)
phpunit/phpunit suggests installing phpunit/php-invoker (^2.0)
Generating optimized autoload files
> @php -r "file_exists('.env') || copy('.env.example', '.env');"
> @php artisan key:generate --ansi
Application key set successfully.
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi
Discovered Package: beyondcode/laravel-dump-server
Discovered Package: fideloper/proxy
Discovered Package: laravel/nexmo-notification-channel
Discovered Package: laravel/slack-notification-channel
Discovered Package: laravel/tinker
Discovered Package: nesbot/carbon
Discovered Package: nunomaduro/collision
Package manifest generated successfully.
Application ready! Build something amazing.
PS C:\xampp\htdocs> _
```

After successfully run the command, a project folder ***task-crud*** will be created under htdocs folder (C:/xampp/htdocs/task-crud). Now in the browser open: `http://localhost/task-crud/public/` you will see



Congratulations!!! You have done.

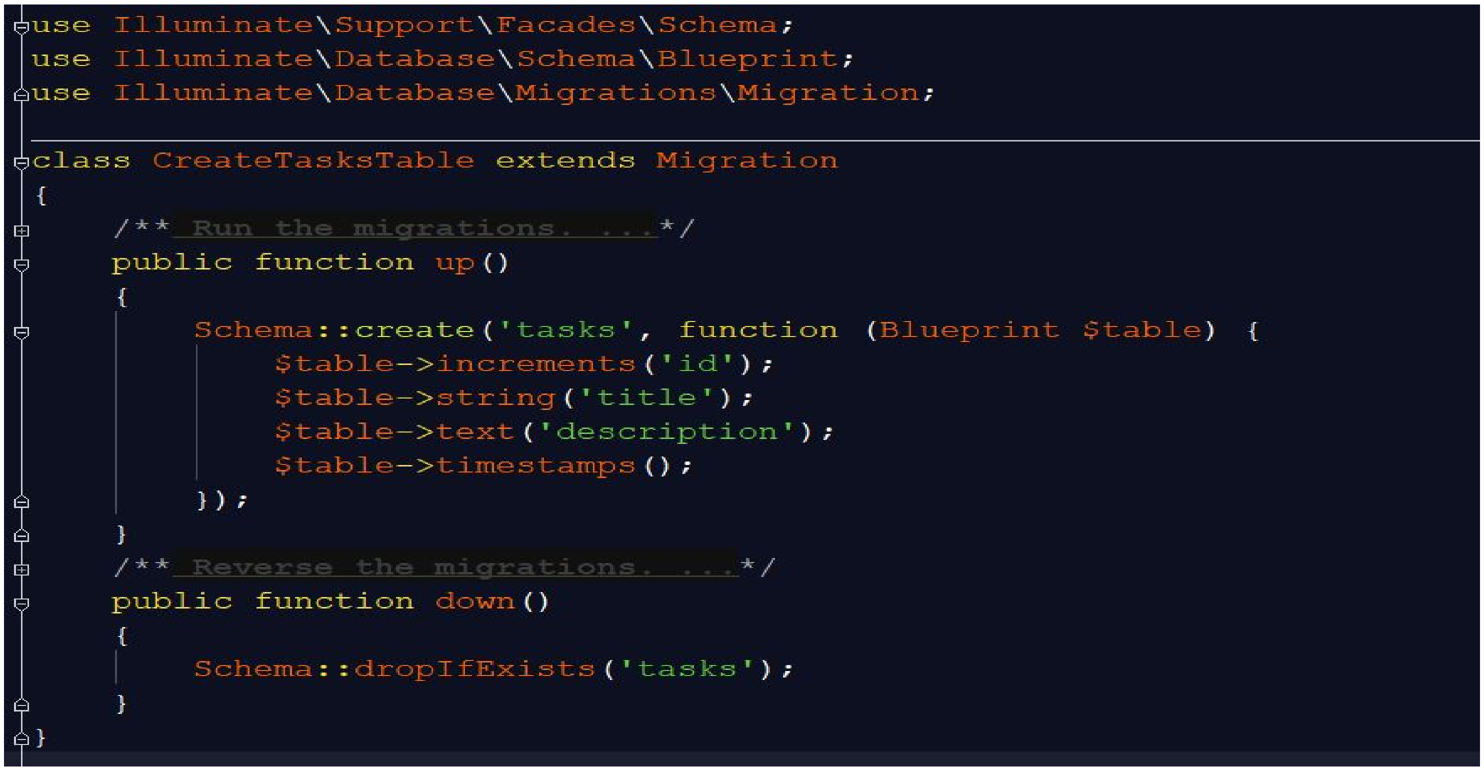
Step 4: Create migration file to create tasks table: In this step we have to create migrations for tasks table using Laravel php artisan command. So let's write the following command: `php artisan make:migration create_tasks_table`



```
C:\xampp\htdocs\task-crud>php artisan make:migration create_tasks_table
Created Migration: 2019_02_07_054748_create_tasks_table
```

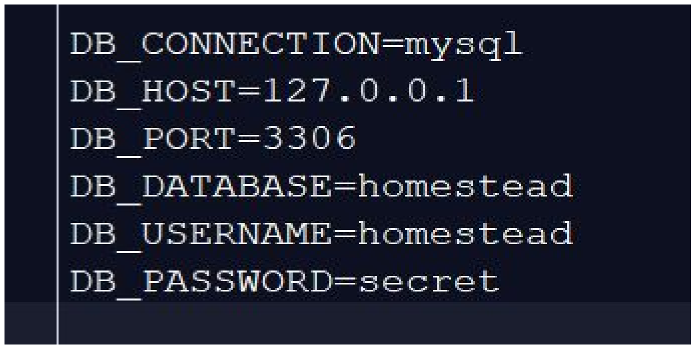
After successfully run the command you will find a file in the following path: ../task-crud/database/migrations 2019_02_07_054748_create_tasks_table.php

Now open the file and add two line ***$table->increments('id'); $table->string('title');*** Like following



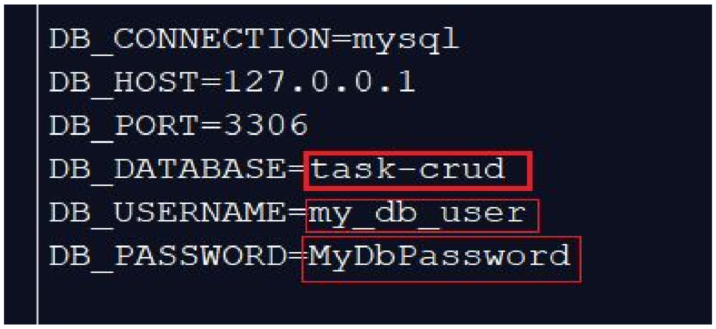
```
use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateTasksTable extends Migration
{
    /** Run the migrations. ...*/
    public function up()
    {
        Schema::create('tasks', function (Blueprint $table) {
            $table->increments('id');
            $table->string('title');
            $table->text('description');
            $table->timestamps();
        });
    }
    /** Reverse the migrations. ...*/
    public function down()
    {
        Schema::dropIfExists('tasks');
    }
}
```

Step 6: Update .env file and Provide Database Credential Open .env file(task-crud/.env)and make necessary changes like the following picture. The credential you need to connect to database.



```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=homestead
DB_USERNAME=homestead
DB_PASSWORD=secret
```

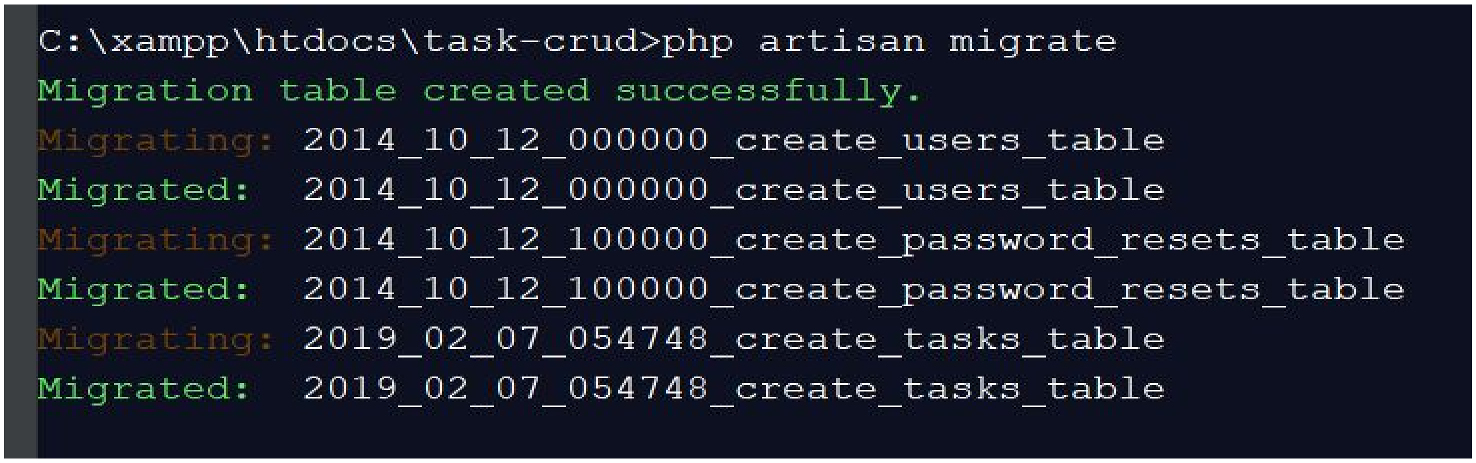
Before modify



```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=task-crud
DB_USERNAME=my_db_user
DB_PASSWORD=MyDbPassword
```

After Modify

Step 7: Create database and Tables Now go to http://localhost/phpmyadmin and create only database ***task-crud.*** Now run the following command `php artisan migrate`



```
C:\xampp\htdocs\task-crud>php artisan migrate
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated:  2014_10_12_000000_create_users_table
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated:  2014_10_12_100000_create_password_resets_table
Migrating: 2019_02_07_054748_create_tasks_table
Migrated:  2019_02_07_054748_create_tasks_table
```

Now check the database in your phpmyadmin. You will find something like this

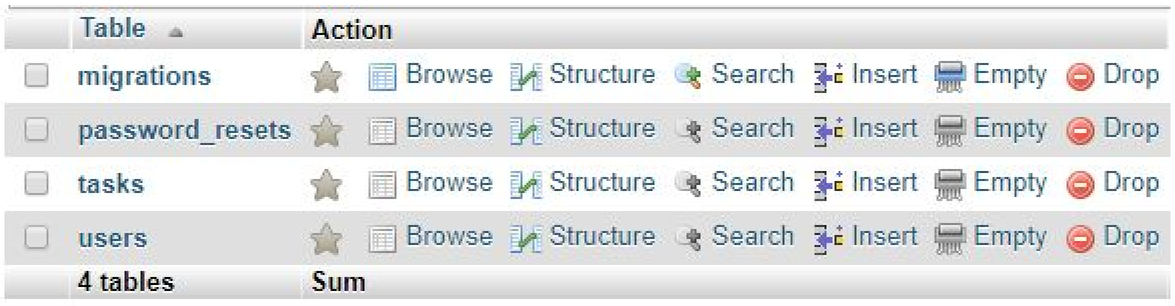


	Table	Action
☐	migrations	☆ Browse Structure Search Insert Empty Drop
☐	password_resets	☆ Browse Structure Search Insert Empty Drop
☐	tasks	☆ Browse Structure Search Insert Empty Drop
☐	users	☆ Browse Structure Search Insert Empty Drop
	4 tables	Sum

Congratulations!!! You have passed another step.

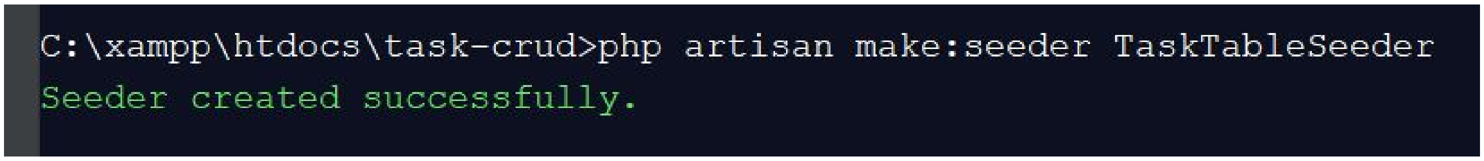
Now we are able to set up or PC for work, Install Laravel and Create Project using Laravel, Create Migration files, Database connection and run Migrate.

In next we will create model and controller to get data from user, insert into database, update record in database delete record from database, view data in a tabular form.

Basic CRUD operation using Laravel Part 2

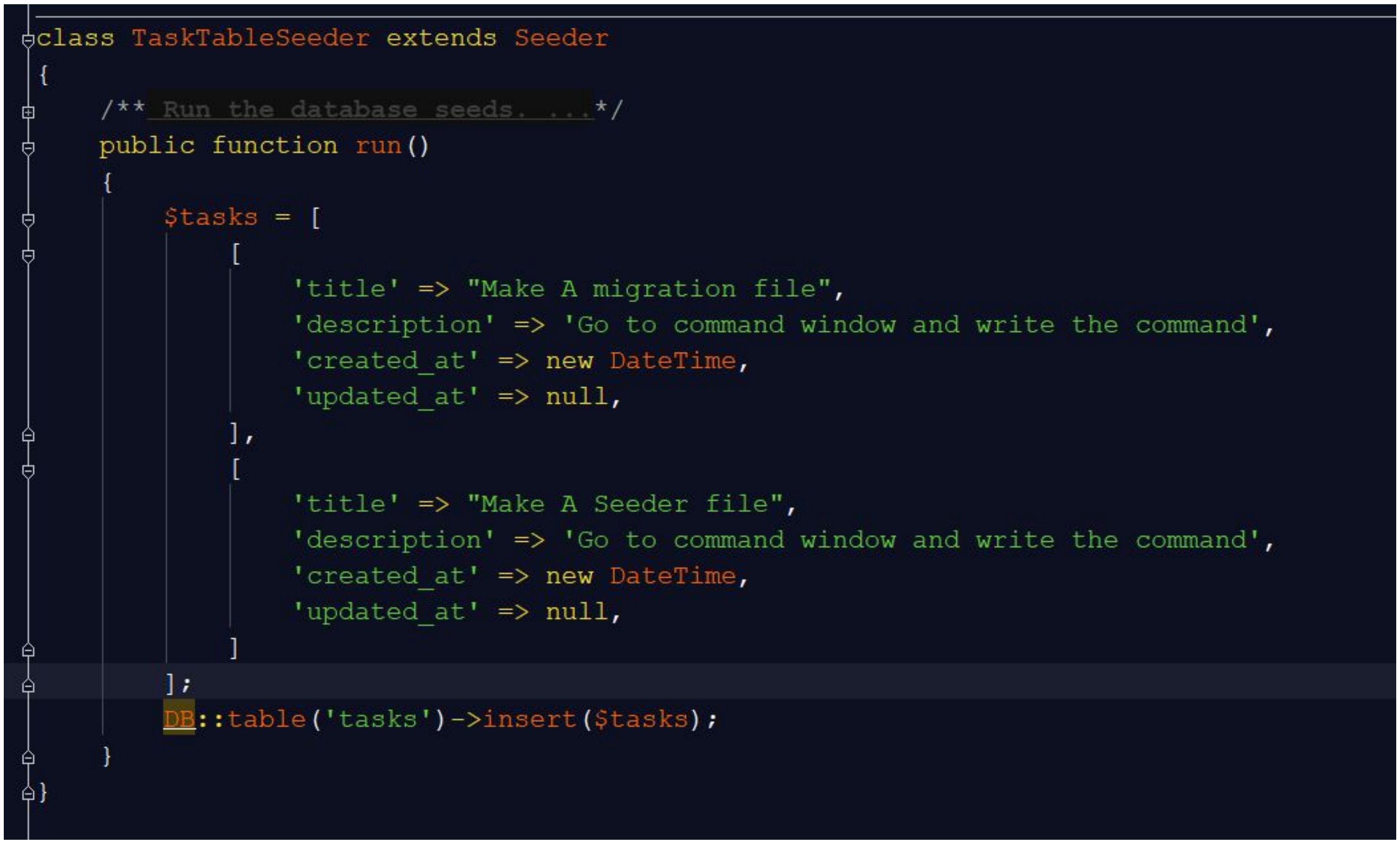
In the previous part we have seen how to install Laravel and create project using it. We have also learned how to create migration file, modifying .env file and DB connection and created database table by migration command. Now we will insert some dummy data, we will view the data in a html table.

Step 1 Create Seeder: Let’s insert some dummy data. To do this, let’s create a seeder file. run the following command `php artisan make:seeder TaskTableSeeder`



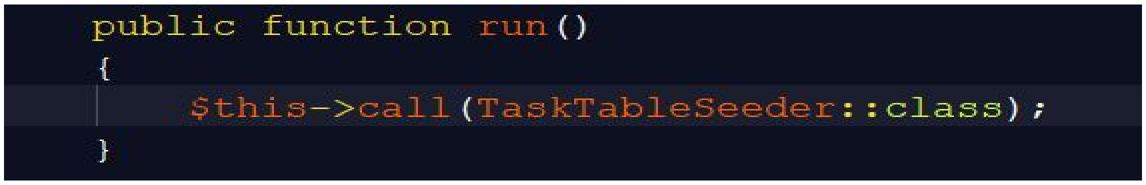
```
C:\xampp\htdocs\task-crud>php artisan make:seeder TaskTableSeeder
Seeder created successfully.
```

After successfully run the command you will find a file in the following path: ../task-crud/database/seeds. Now open and modify like following



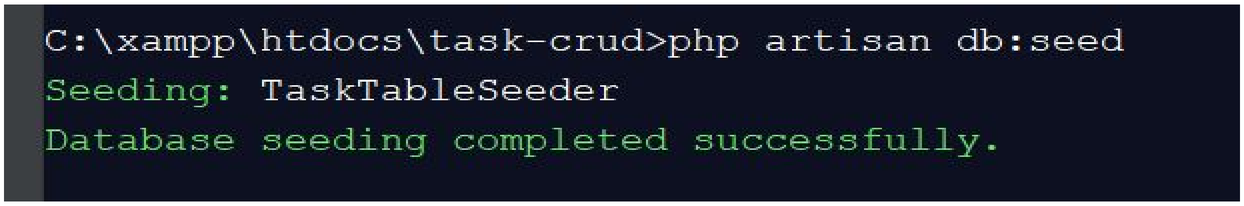
```
class TaskTableSeeder extends Seeder
{
    /** Run the database seeds. ...*/
    public function run()
    {
        $tasks = [
            [
                'title' => "Make A migration file",
                'description' => 'Go to command window and write the command',
                'created_at' => new DateTime,
                'updated_at' => null,
            ],
            [
                'title' => "Make A Seeder file",
                'description' => 'Go to command window and write the command',
                'created_at' => new DateTime,
                'updated_at' => null,
            ]
        ];
        DB::table('tasks')->insert($tasks);
    }
}
```

Now open the (../task-crud/database/seeds/DatabaseSeeder.php) file and update like following:



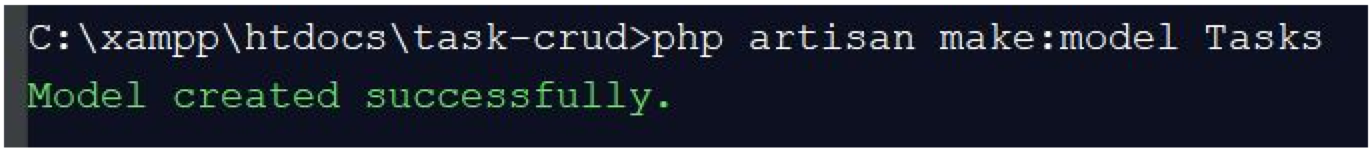
```
    public function run()
    {
        $this->call(TaskTableSeeder::class);
    }
```

Now run the following command: `php artisan db:seed` After successfully run the command you can check your database. You will find some data into tasks table.



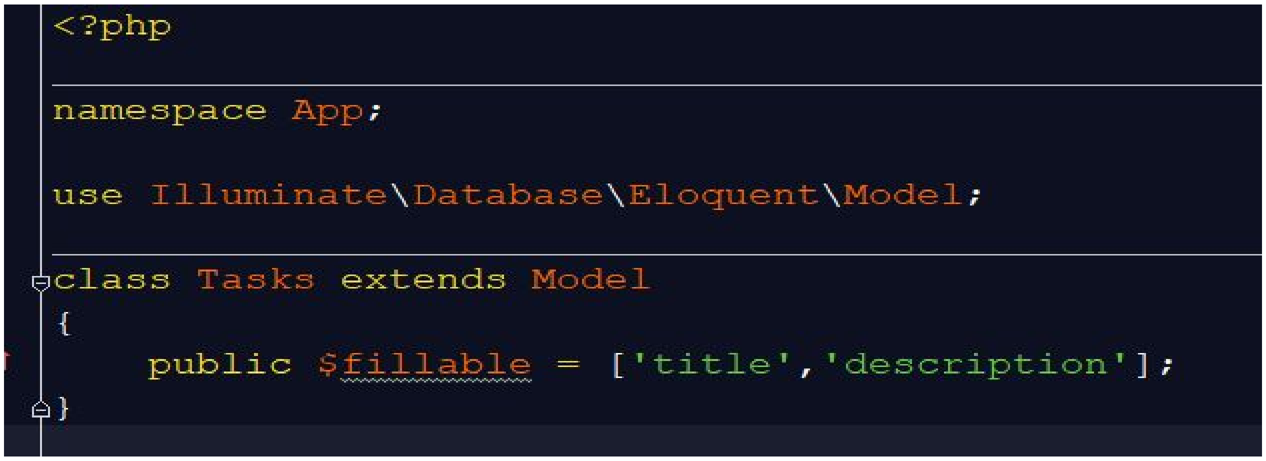
```
C:\xampp\htdocs\task-crud>php artisan db:seed
Seeding: TaskTableSeeder
Database seeding completed successfully.
```

Step 2 Create Task Modal: In this step we will run the following command `php artisan make:modal Tasks`



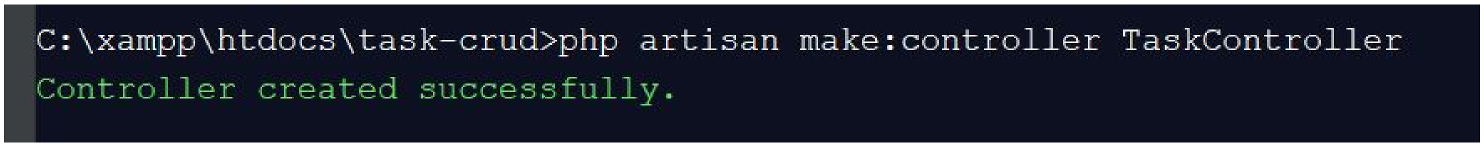
```
C:\xampp\htdocs\task-crud>php artisan make:model Tasks
Model created successfully.
```

After successfully run the command you will find a file in the following path: ../task-crud/app/tasks.php Now open the file and add one line public $fillable = ['title','description']; like following



```php
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Tasks extends Model
{
    public $fillable = ['title','description'];
}
```

Step 3 Create Task Controller: In this step we will run the following command `php artisan make:controller TaskController`

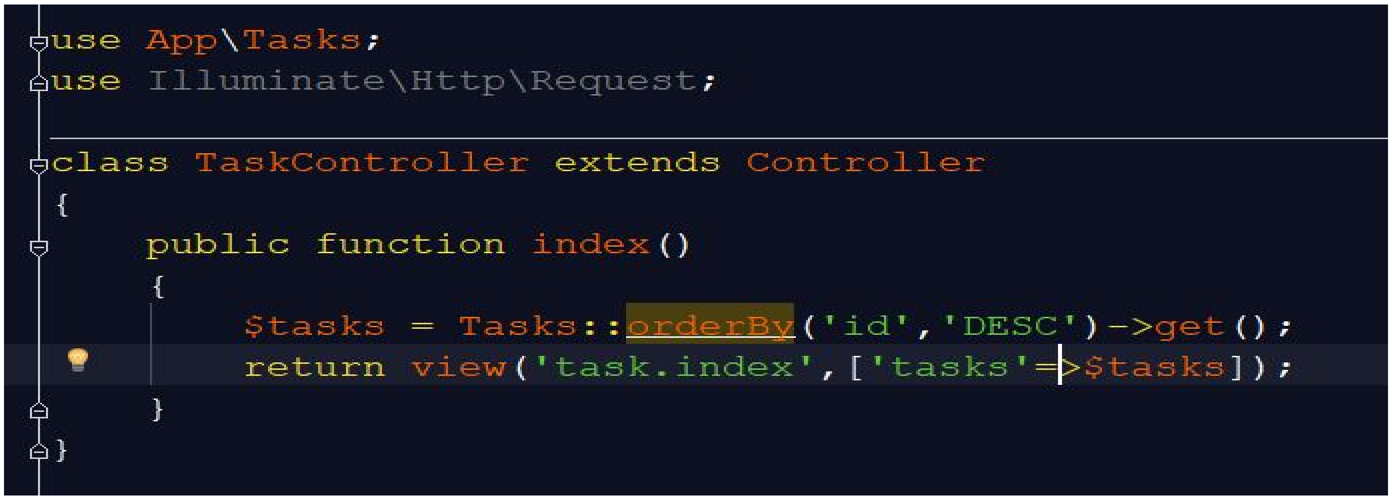


```
C:\xampp\htdocs\task-crud>php artisan make:controller TaskController
Controller created successfully.
```

After successfully run the command you will find a file in the following path: ../task-crud/app/http/ This controller will manage all route method. On another word I can say, we will write all the function to take care of the request sent from browser and provide response. Let's see what type of request we are expecting

1. localhost/task-crud/public/tasks: (GET method to get the list of tasks with html table)
2. localhost/task-crud/public/tasks/create (GET method to get a html form to create task)
3. localhost/task-crud/public/tasks/insert (POST method to submit form and insert into database.)
4. localhost/task-crud/public/tasks/update?id=someValue (GET method to get a update / edit form)
5. localhost/task-crud/public/tasks/insert (POST/PUT method to submit form and update into database.)
6. localhost/task-crud/public/tasks/update?id=someValue (POST/DELETE method to delete record)

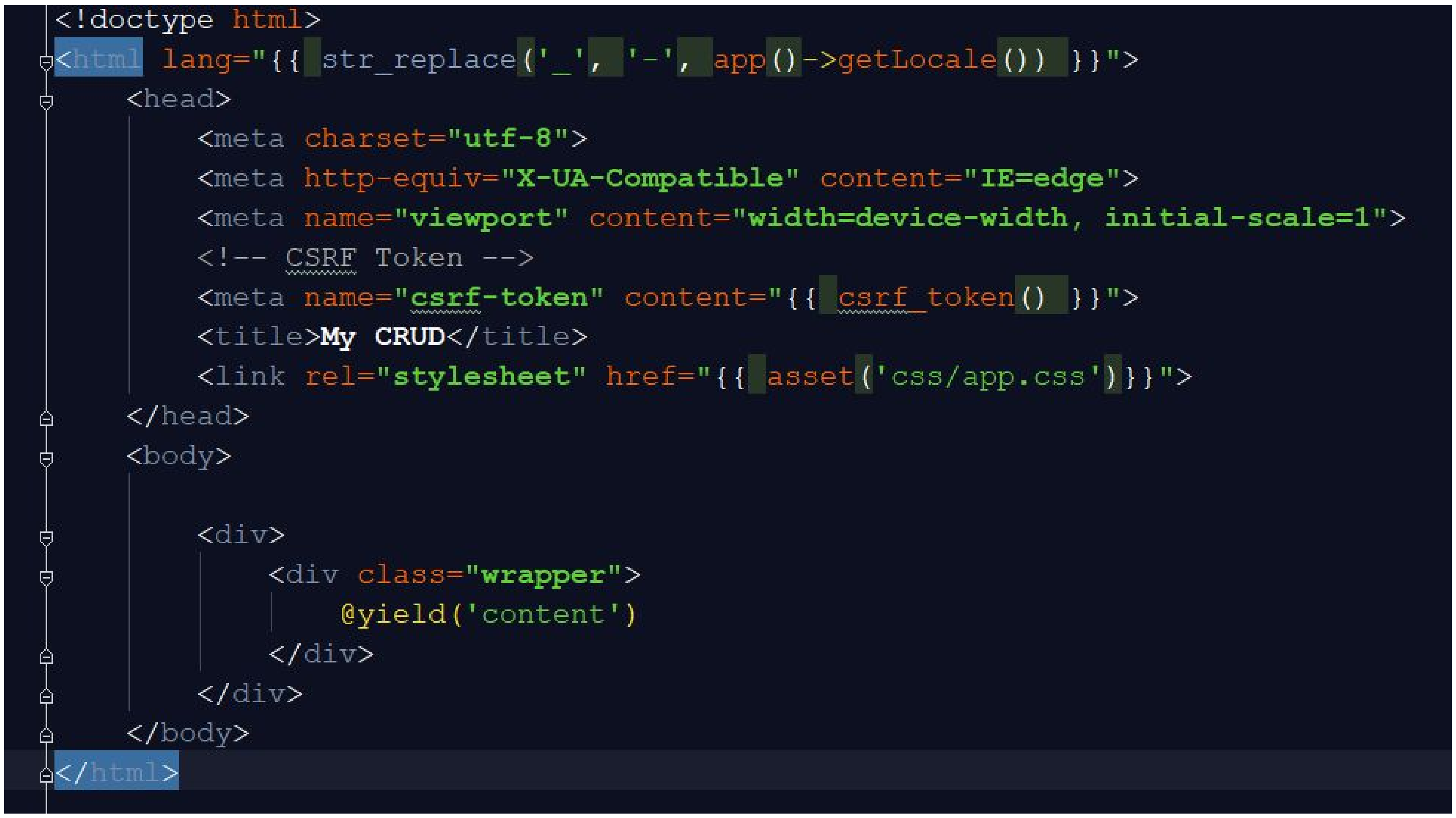
Process Request 1: Let’s open TaskController and modify like following



```php
use App\Tasks;
use Illuminate\Http\Request;

class TaskController extends Controller
{
    public function index()
    {
        $tasks = Tasks::orderBy('id','DESC')->get();
        return view('task.index',['tasks'=>$tasks]);
    }
}
```

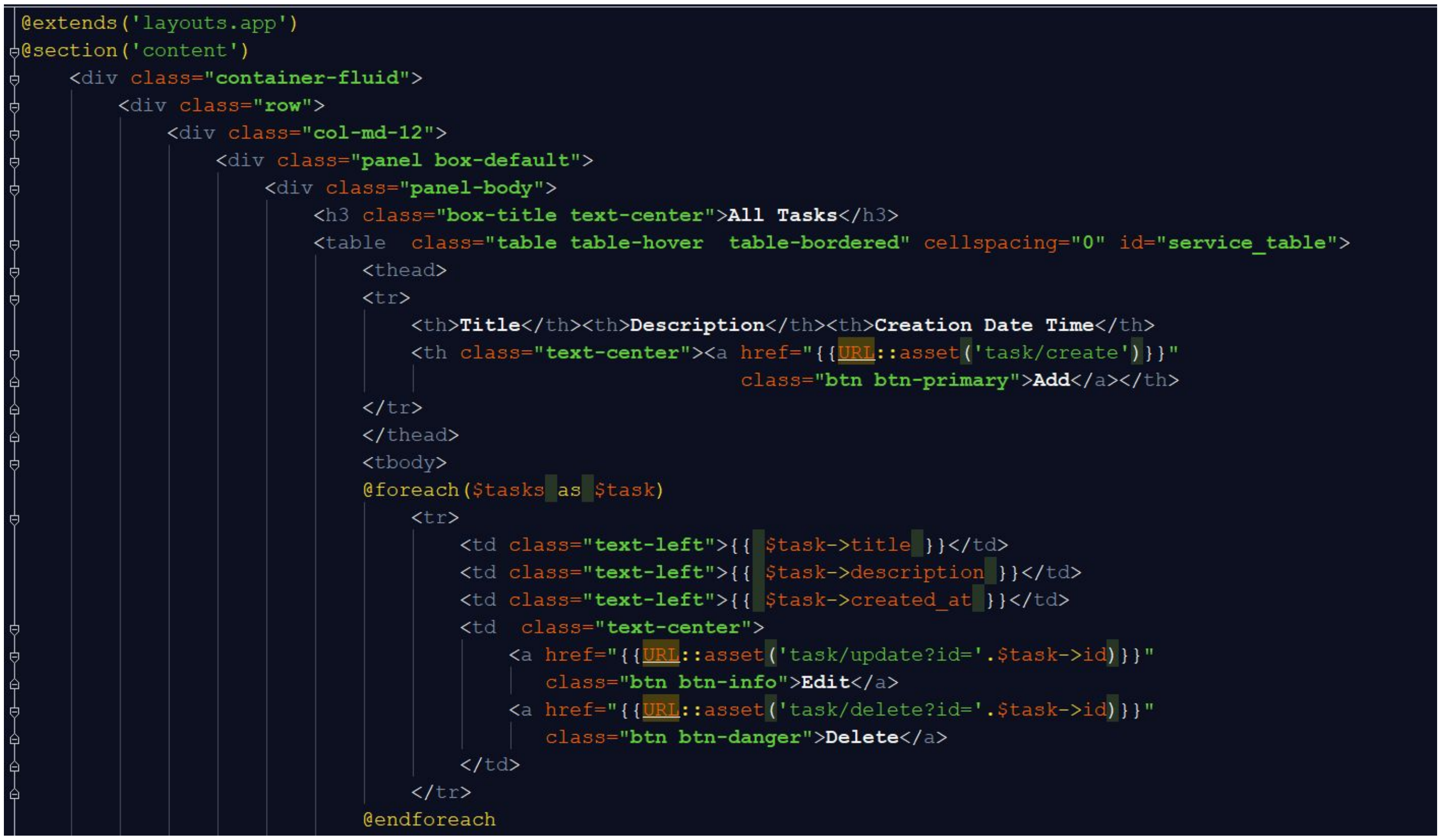
In the above code we are running one query and fetch all tasks from tasks table and putted data into $tasks variable. In the next line we are calling one view file index.blade.php and passing the data using tasks variable.

Step 4 Work with View: Let’s create a folder (***layouts***) inside ../task-crud/resources/views. Now inside ***layout (***../task-crud/resources/views/layout***)*** folder create a blade file (***app.blade.php***). Open the file and modify like following

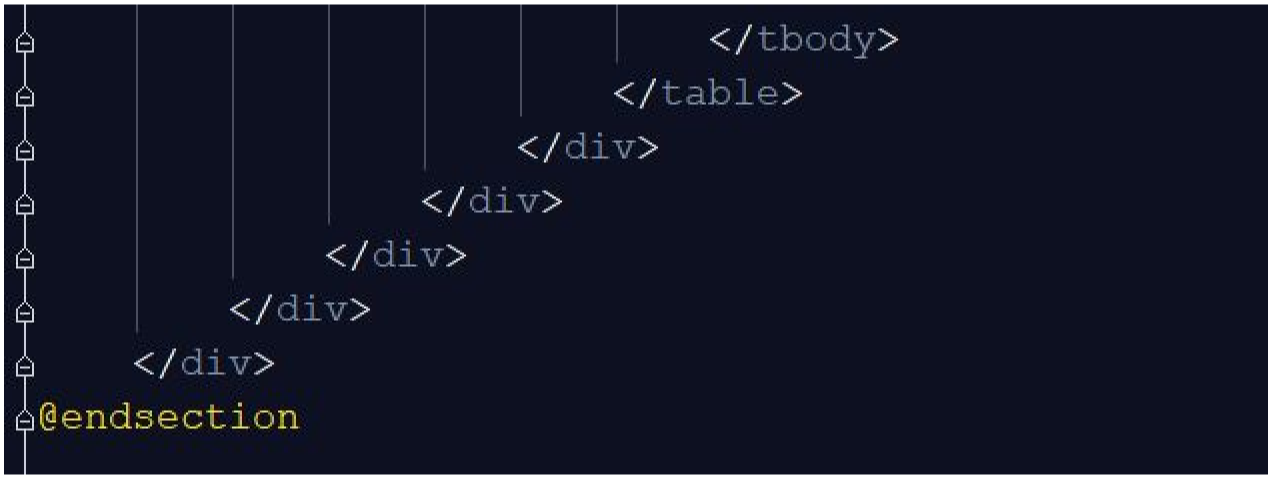


```html
<!doctype html>
<html lang="{{ str_replace('_', '-', app()->getLocale()) }}">
    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <!-- CSRF Token -->
        <meta name="csrf-token" content="{{ csrf_token() }}">
        <title>My CRUD</title>
        <link rel="stylesheet" href="{{ asset('css/app.css')}}">
    </head>
    <body>

        <div>
            <div class="wrapper">
                @yield('content')
            </div>
        </div>
    </body>
</html>
```

Let’s create another folder (***task***) inside ../task-crud/resources/views. Now inside ***task(***../task-crud/resources/views/task***)*** folder create a blade file (***index.blade.php***). Open the file and modify like following

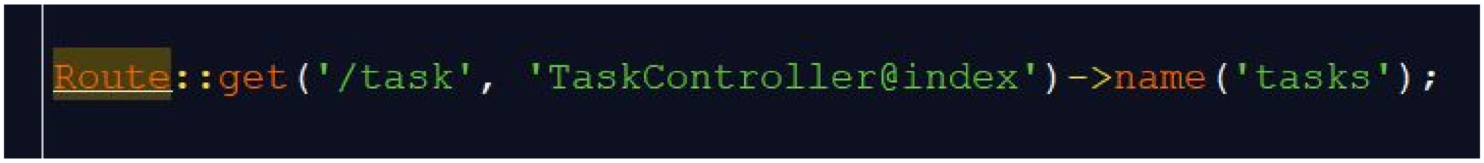


```
@extends('layouts.app')
@section('content')
    <div class="container-fluid">
        <div class="row">
            <div class="col-md-12">
                <div class="panel box-default">
                    <div class="panel-body">
                        <h3 class="box-title text-center">All Tasks</h3>
                        <table  class="table table-hover  table-bordered" cellspacing="0" id="service_table">
                            <thead>
                            <tr>
                                <th>Title</th><th>Description</th><th>Creation Date Time</th>
                                <th class="text-center"><a href="{{URL::asset('task/create')}}"
                                                           class="btn btn-primary">Add</a></th>
                            </tr>
                            </thead>
                            <tbody>
                            @foreach($tasks as $task)
                                <tr>
                                    <td class="text-left">{{ $task->title }}</td>
                                    <td class="text-left">{{ $task->description }}</td>
                                    <td class="text-left">{{ $task->created_at }}</td>
                                    <td  class="text-center">
                                        <a href="{{URL::asset('task/update?id='.$task->id)}}"
                                           class="btn btn-info">Edit</a>
                                        <a href="{{URL::asset('task/delete?id='.$task->id)}}"
                                           class="btn btn-danger">Delete</a>
                                    </td>
                                </tr>
                            @endforeach
```



```
                            </tbody>
                        </table>
                    </div>
                </div>
            </div>
        </div>
    </div>
@endsection
```

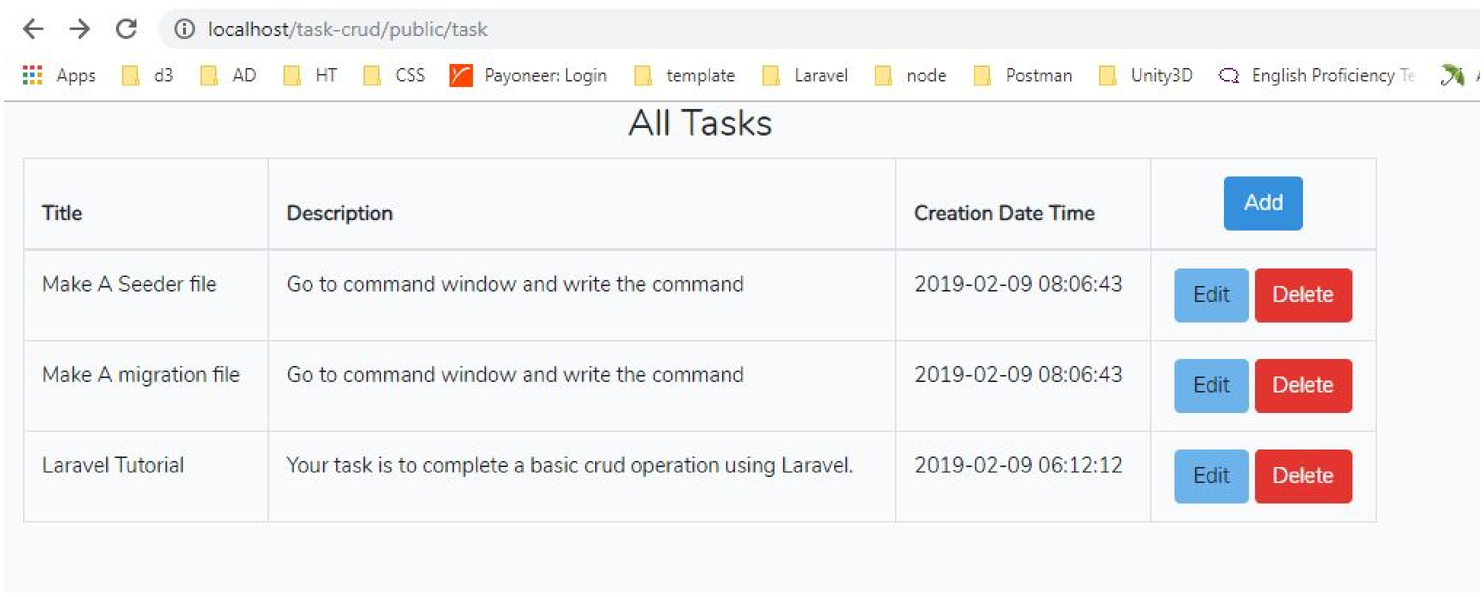
Step 4 Work with Route: Go you your project repository and find the routes folder and open web.php file (../task-crud/routes/web.php). Modify the file like following



```
Route::get('/task', 'TaskController@index')->name('tasks');
```

Here we are defining, if the url is localhost/task-crud/public/task then we will call the index function of TaskController.php. We also define the method as GET.

Now in the browser open: `http://localhost/task-crud/public/task` you will see



All Tasks

Title	Description	Creation Date Time	Add
Make A Seeder file	Go to command window and write the command	2019-02-09 08:06:43	Edit Delete
Make A migration file	Go to command window and write the command	2019-02-09 08:06:43	Edit Delete
Laravel Tutorial	Your task is to complete a basic crud operation using Laravel.	2019-02-09 06:12:12	Edit Delete

Congratulations!!! You have passed another step.

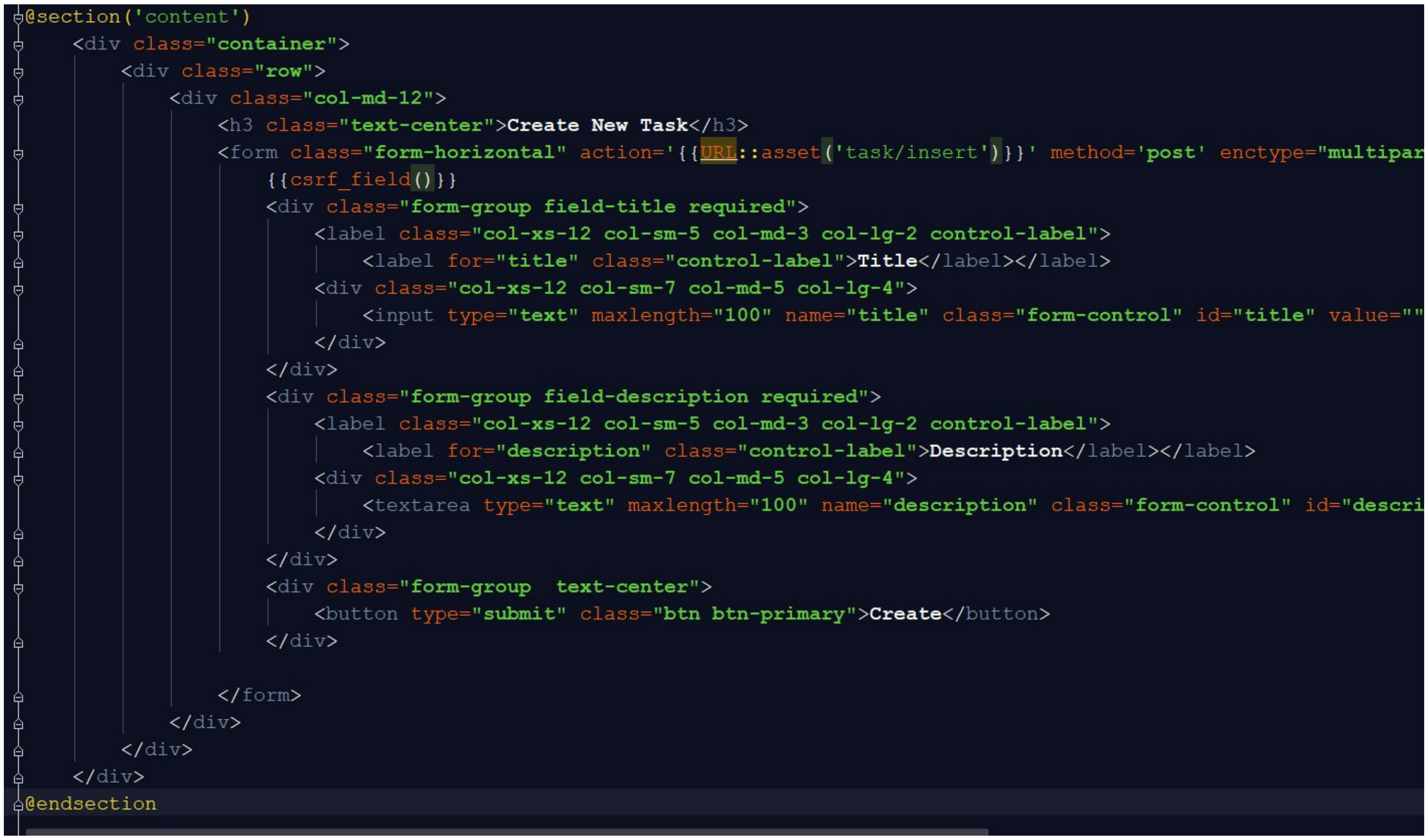
Basic CRUD operation using Laravel Part 3

In the previous part we have created seeder file, run seeder, created controller, written index method inside controller, modified routes for get index function, created few view files and called them and return a html tabular view inside web page. In this part we will create views for create form, update form. Also we will write method for call others requests. So let's start

In the previous part we have shown a html table. In the action column of the Table there is a ***Add*** button. When user will click on that button System will send a request to server and server will receive this request and generate a html form and response to that request. http://localhost/task-crud/public/task/create . For this reason we will create a blade file to generate html form and a function(***create_form***) in our TaskController to call that blade file.

Step 1: Create a blade file

Go to ***task(***../task-crud/resources/views/task***)*** folder create a blade file (***create_form.blade.php***). Open the file and modify like following



```
@section('content')
    <div class="container">
        <div class="row">
            <div class="col-md-12">
                <h3 class="text-center">Create New Task</h3>
                <form class="form-horizontal" action='{{URL::asset('task/insert')}}' method='post' enctype="multipar
                    {{csrf_field()}}
                    <div class="form-group field-title required">
                        <label class="col-xs-12 col-sm-5 col-md-3 col-lg-2 control-label">
                            <label for="title" class="control-label">Title</label></label>
                        <div class="col-xs-12 col-sm-7 col-md-5 col-lg-4">
                            <input type="text" maxlength="100" name="title" class="form-control" id="title" value=""
                        </div>
                    </div>
                    <div class="form-group field-description required">
                        <label class="col-xs-12 col-sm-5 col-md-3 col-lg-2 control-label">
                            <label for="description" class="control-label">Description</label></label>
                        <div class="col-xs-12 col-sm-7 col-md-5 col-lg-4">
                            <textarea type="text" maxlength="100" name="description" class="form-control" id="descri
                        </div>
                    </div>
                    <div class="form-group  text-center">
                        <button type="submit" class="btn btn-primary">Create</button>
                    </div>

                </form>
            </div>
        </div>
    </div>
@endsection
```

Step 2: Add function in controller. Now open TaskController and create a function(***create_form***) to view an html form to the browser. Modify like following:

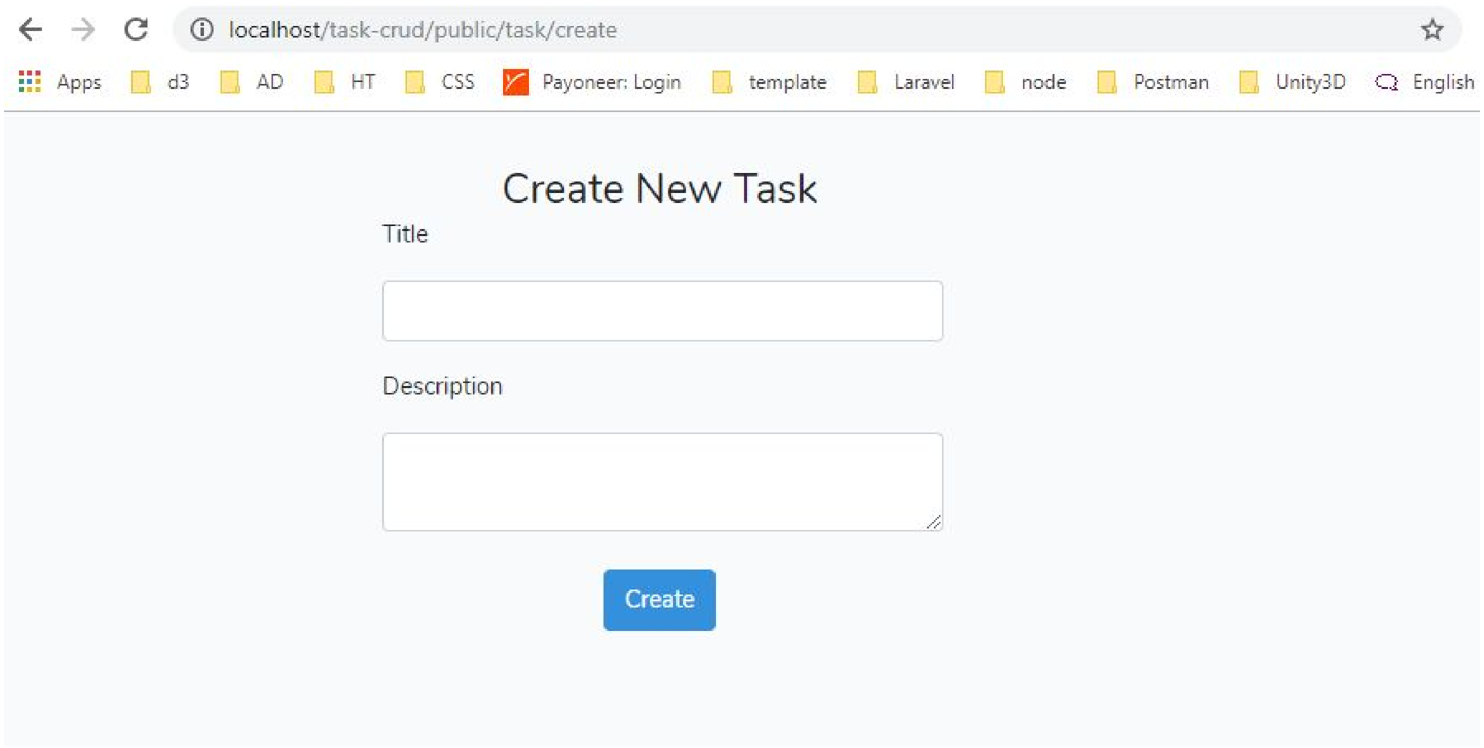
```
    public function create_form()
    {
        return view('task.create_form');
    }
```

Step 3: Modify Route File

Let’s open web.php file (../task-crud/routes/web.php). And add another line to response this request. Like following:

```
Route::get('/task', 'TaskController@index');
Route::get('/task/create', 'TaskController@create_form');
```

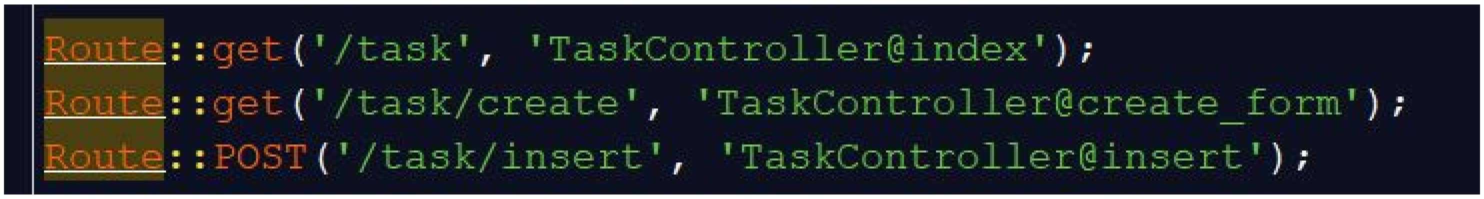
Now in the browser open: `http://localhost/task-crud/public/task` Now click on ***Add*** button you will see like following:-



Congratulations!!! You have passed another step.Now we can show a form. Now when user will fill up the form and click on ***Create*** button, system will send a request to server with form data with post method. Server will receive the request, process data and insert into database, then show the table with a success message. To do this we will create another method(***insert***) in our TaskController. Let’s open TaskController and create a function(***insert***) and modify like following:

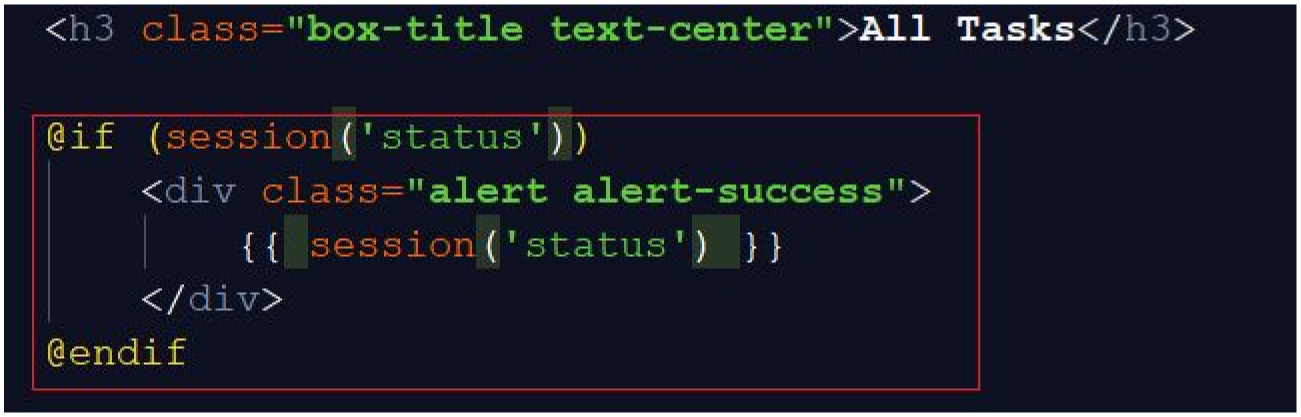
```
public function insert(Request $request){
    Tasks::create($request->only('title','description'));
    return redirect()->action('TaskController@index')->with('status', 'Task Created Successfully');
}
```

Also add another line to response this request. Like following:



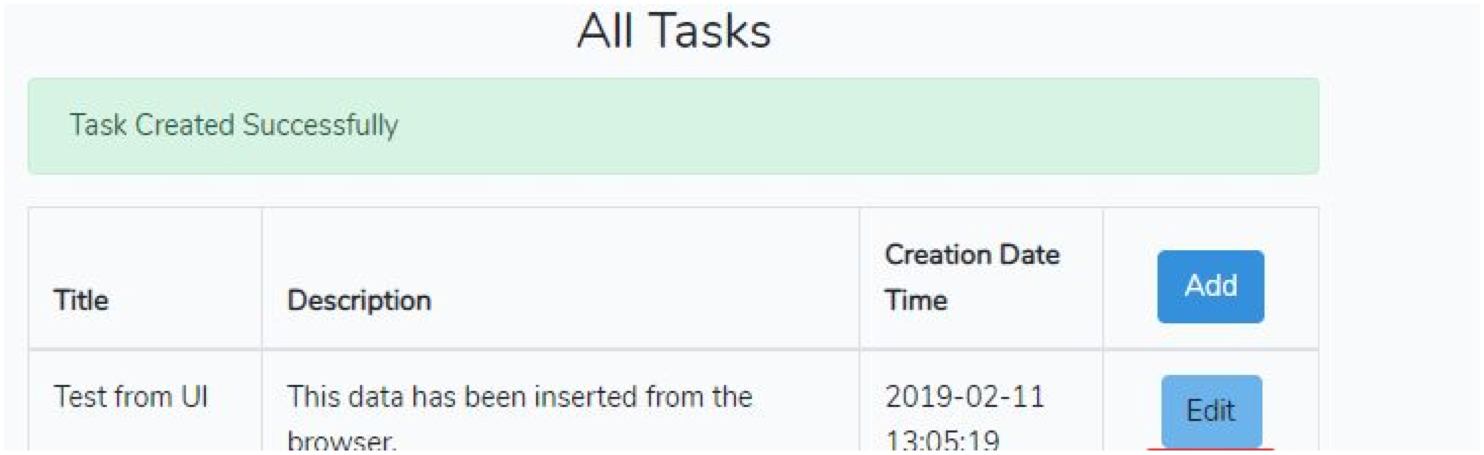
```
Route::get('/task', 'TaskController@index');
Route::get('/task/create', 'TaskController@create_form');
Route::POST('/task/insert', 'TaskController@insert');
```

Let's modify our index.blade.php to show the success message



```
<h3 class="box-title text-center">All Tasks</h3>

@if (session('status'))
    <div class="alert alert-success">
        {{ session('status') }}
    </div>
@endif
```

Now if you fill up the form and click on Create button you will see something like following:



All Tasks

Task Created Successfully

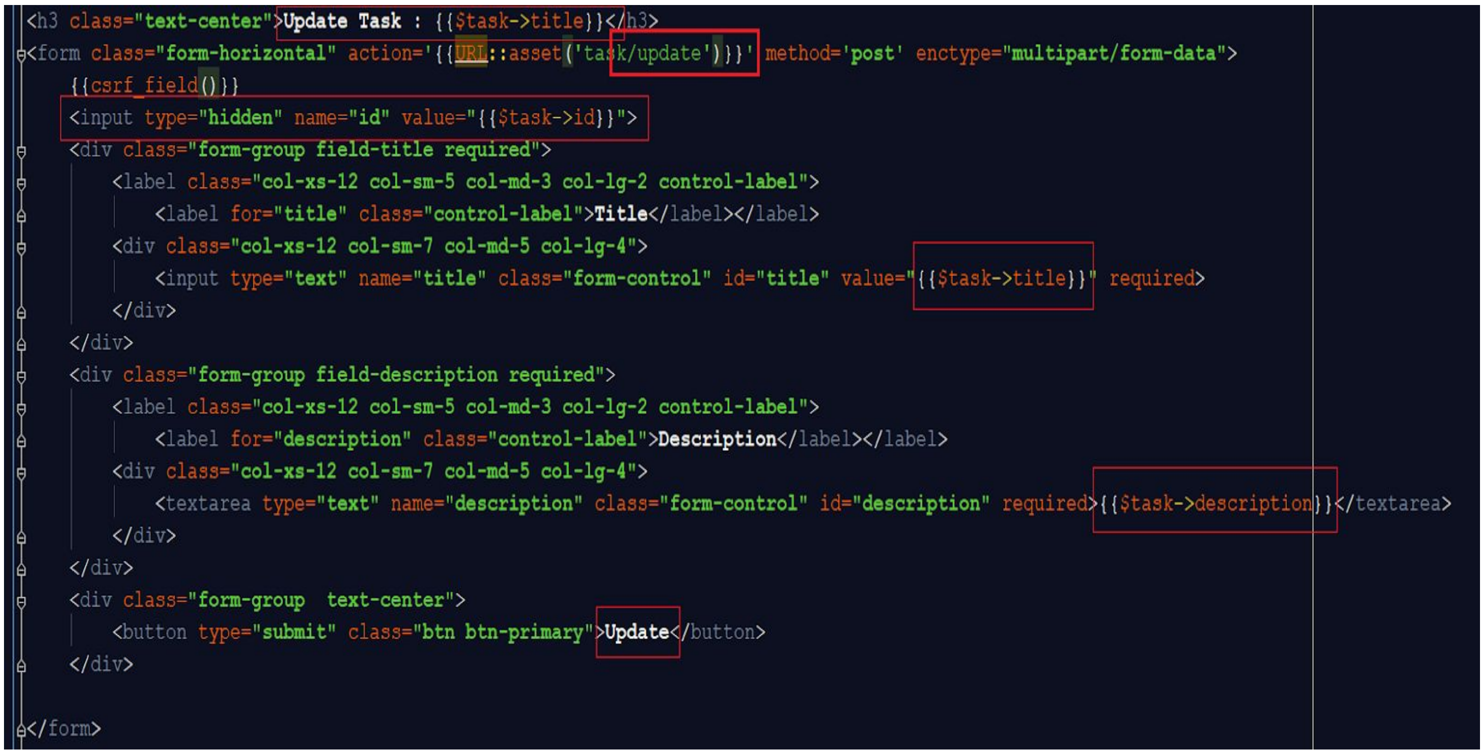
Title	Description	Creation Date Time	Add
Test from UI	This data has been inserted from the browser.	2019-02-11 13:05:19	Edit

Congratulations!!! You have passed another step.

In each row of the table action column contains two button one is ***Edit*** and another is ***Delete***. When user will click on ***Edit*** button, System will send a request with table id to server and server will receive that request and generate a html form with existing data for the id passed during request and response to that request. When user click on ***Delete*** button, System will send a request with table id to server and server will receive that request and delete the row for the id passed during request and response to that request. Edit request will be something like http://localhost/task-crud/public/task/update?id=10 Delete request will be something like http://localhost/task-crud/public/task/delete?id=10 For Edit request we will create a blade file to generate html form and a function(***update_form***) in our TaskController to call that blade file.

Step 1: Create a blade file

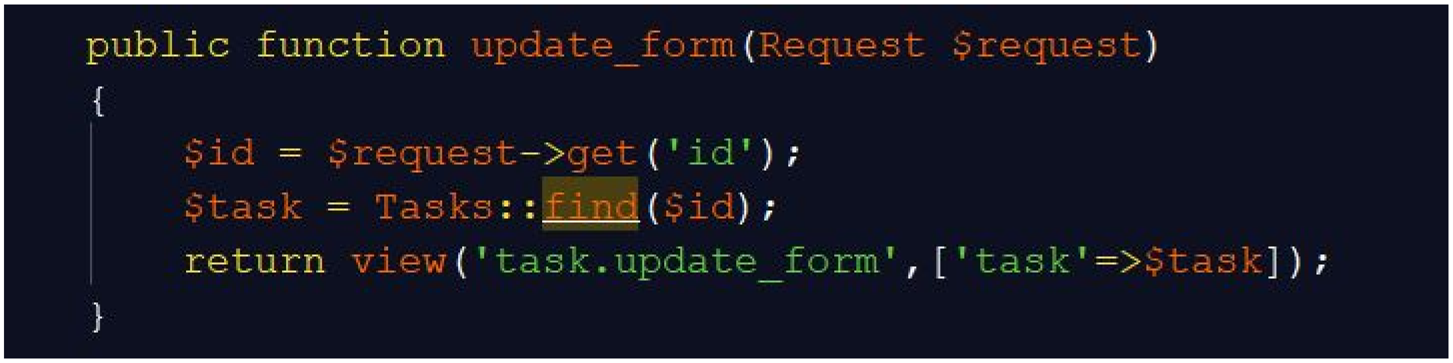
Go to ***task(***../task-crud/resources/views/task***)*** folder create a blade file (***update_form.blade.php***). Open the file and modify like following



```
<h3 class="text-center">Update Task : {{$task->title}}</h3>
<form class="form-horizontal" action='{{URL::asset('task/update')}}' method='post' enctype="multipart/form-data">
    {{csrf_field()}}
    <input type="hidden" name="id" value="{{$task->id}}">
    <div class="form-group field-title required">
        <label class="col-xs-12 col-sm-5 col-md-3 col-lg-2 control-label">
            <label for="title" class="control-label">Title</label></label>
        <div class="col-xs-12 col-sm-7 col-md-5 col-lg-4">
            <input type="text" name="title" class="form-control" id="title" value="{{$task->title}}" required>
        </div>
    </div>
    <div class="form-group field-description required">
        <label class="col-xs-12 col-sm-5 col-md-3 col-lg-2 control-label">
            <label for="description" class="control-label">Description</label></label>
        <div class="col-xs-12 col-sm-7 col-md-5 col-lg-4">
            <textarea type="text" name="description" class="form-control" id="description" required>{{$task->description}}</textarea>
        </div>
    </div>
    <div class="form-group  text-center">
        <button type="submit" class="btn btn-primary">Update</button>
    </div>

</form>
```

Note: *Here the red block is the only difference between create_form.blade.php and update_form.blade.php*

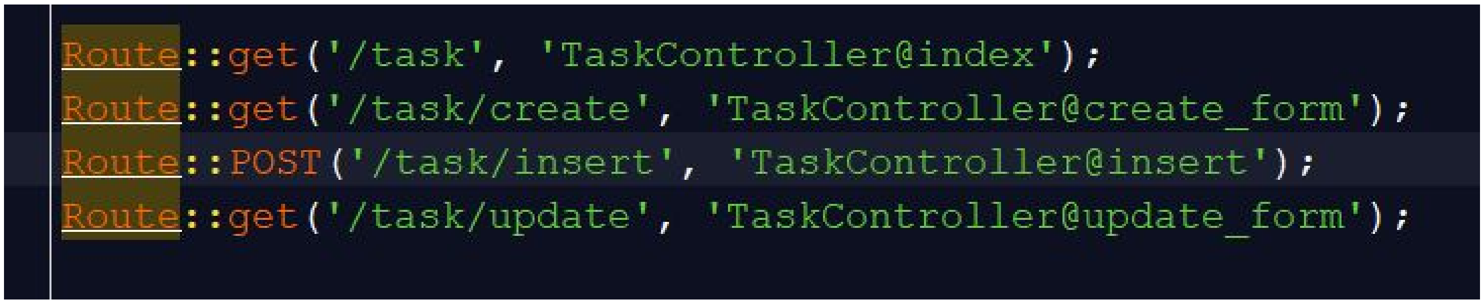
Step 2: Add function in controller. Now open TaskController and create a function(***update_form***). Modify like following:



```
public function update_form(Request $request)
{
    $id = $request->get('id');
    $task = Tasks::find($id);
    return view('task.update_form',['task'=>$task]);
}
```

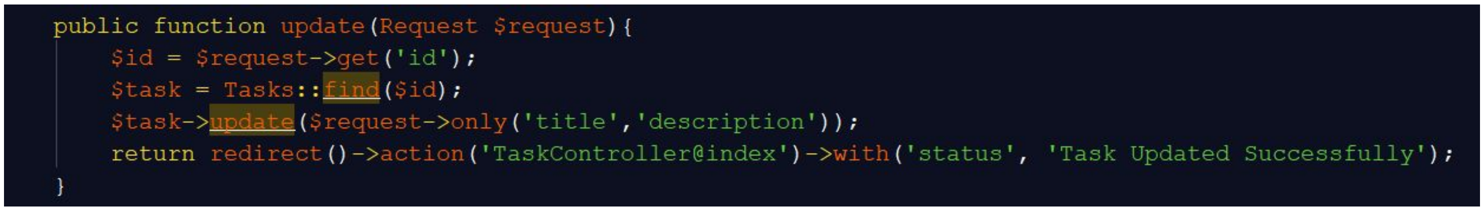
Step 3: Modify Route File

Let's open web.php file (../task-crud/routes/web.php). And add another line to response this request. Like following:



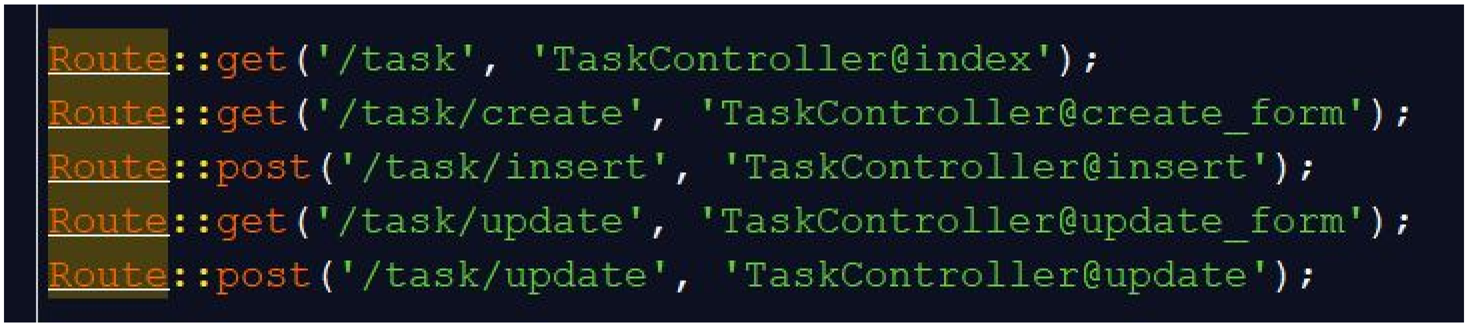
```
Route::get('/task', 'TaskController@index');
Route::get('/task/create', 'TaskController@create_form');
Route::POST('/task/insert', 'TaskController@insert');
Route::get('/task/update', 'TaskController@update_form');
```

Now in the browser open: `http://localhost/task-crud/public/task` Now click on ***Edit*** button you will see an update form with existing value. **Congratulations!!!** You have passed another step.Now we can show a form with existing value. Now when user make necessary changes and click on ***Update*** button, system will send a request to server with form data with post method. Server will receive the request, process data and update into database, then show the table with a success message. To do this we will create another method(***update***) in our TaskController. Let's open TaskController and create a function(***update***) and modify like following:



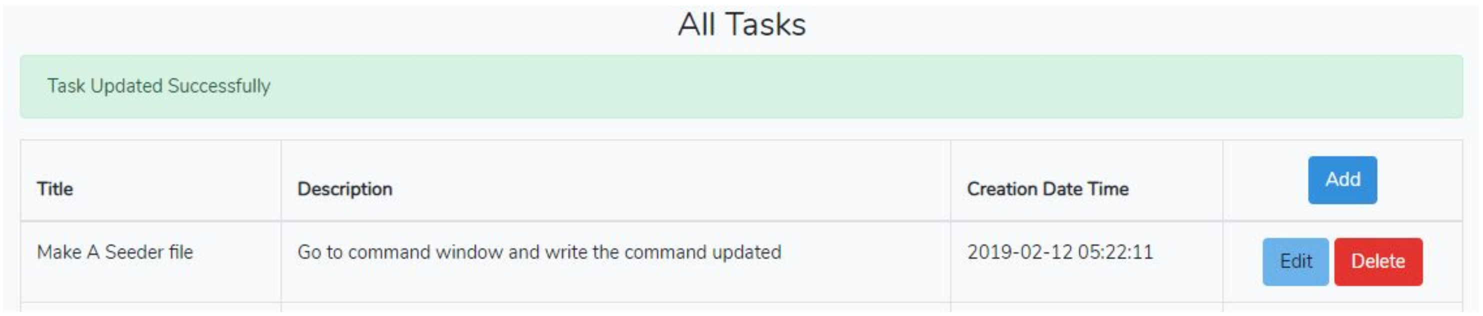
```
public function update(Request $request){
    $id = $request->get('id');
    $task = Tasks::find($id);
    $task->update($request->only('title','description'));
    return redirect()->action('TaskController@index')->with('status', 'Task Updated Successfully');
}
```

Also add another line in web.php file to response this request. Like following:



```
Route::get('/task', 'TaskController@index');
Route::get('/task/create', 'TaskController@create_form');
Route::post('/task/insert', 'TaskController@insert');
Route::get('/task/update', 'TaskController@update_form');
Route::post('/task/update', 'TaskController@update');
```

Now if you click on ***Update*** button in the form, you will see something like following:-

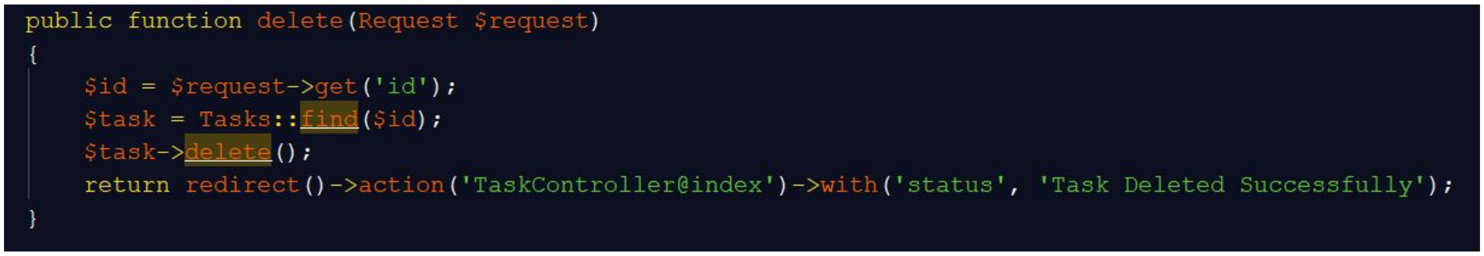


All Tasks

Task Updated Successfully

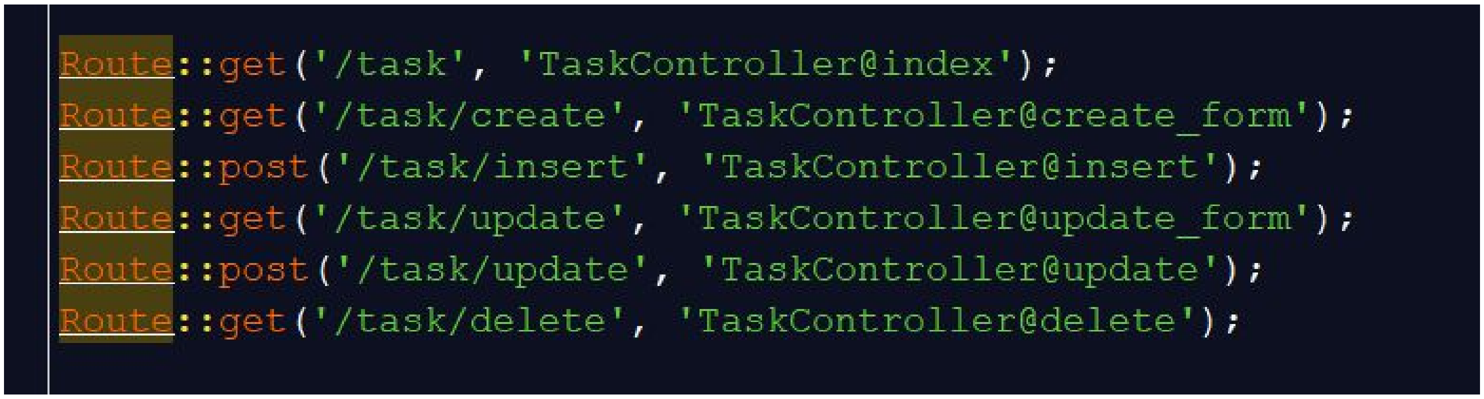
Title	Description	Creation Date Time	Add
Make A Seeder file	Go to command window and write the command updated	2019-02-12 05:22:11	Edit Delete

Congratulations!!! You have passed another step. For delete request we will only add one function to our controller to delete that record and we will show updated table to user. To do this let's open TaskController and create a function(***delete***) and modify like following:



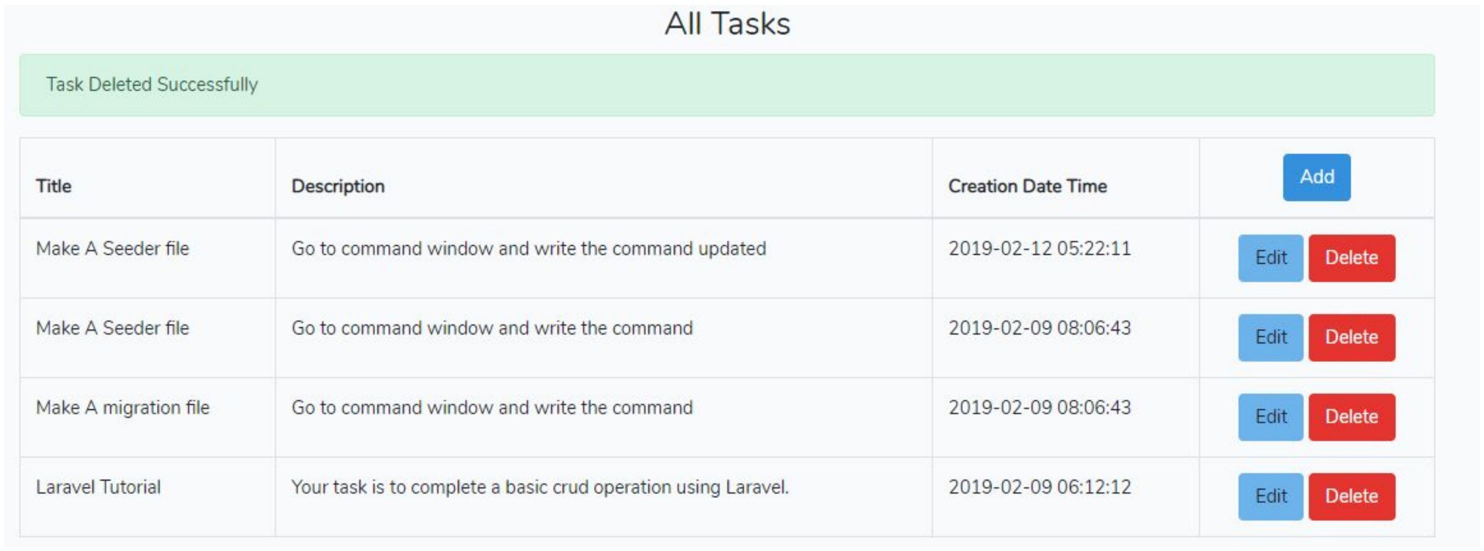
```
public function delete(Request $request)
{
    $id = $request->get('id');
    $task = Tasks::find($id);
    $task->delete();
    return redirect()->action('TaskController@index')->with('status', 'Task Deleted Successfully');
}
```

Also add another line in web.php file to response this request. Like following:



```
Route::get('/task', 'TaskController@index');
Route::get('/task/create', 'TaskController@create_form');
Route::post('/task/insert', 'TaskController@insert');
Route::get('/task/update', 'TaskController@update_form');
Route::post('/task/update', 'TaskController@update');
Route::get('/task/delete', 'TaskController@delete');
```

Now if you click on ***Delete*** button in the table, you will see something like following:-



All Tasks

Task Deleted Successfully

Title	Description	Creation Date Time	Add
Make A Seeder file	Go to command window and write the command updated	2019-02-12 05:22:11	Edit Delete
Make A Seeder file	Go to command window and write the command	2019-02-09 08:06:43	Edit Delete
Make A migration file	Go to command window and write the command	2019-02-09 08:06:43	Edit Delete
Laravel Tutorial	Your task is to complete a basic crud operation using Laravel.	2019-02-09 06:12:12	Edit Delete

Congratulations!!! You have passed another step and this is the final step.

As title says it was a basic CRUD operation. Still we have chances to improve. However I hope you enjoyed it.