TP-15

PHP, Laravel

Laravel – Short Introduction

Laravel is a PHP based web framework for building high-end web applications using its significant and graceful syntaxes.

Some Facts About Laravel

- **Taylor Otwell** developed Laravel in **July 2011**, and it was released more than five years after the release of the Codeigniter framework.
- Laravel is a PHP based web-framework like Codeigniter.
- Laravel is one of the open-source PHP frameworks.
- Laravel follows the model-view-controller (MVC) architectural pattern.
- Laravel is one of the most popular PHP frameworks after Codeigniter

Features of Laravel

- Routing controllers
- Configuration management
- Testability
- Authentication and authorization of users
- Modularity
- ORM (Object Relational Mapper) features
- Provides a template engine
- Building schemas
- E-mailing facilities

Laravel – Short Introduction

Laravel Release History

Version	Released on
Laravel 1	June 9, 2011
Laravel 2	November 24, 2011
Laravel 3	February 22, 2012
Laravel 4	May 28, 2013
Laravel 5	February 2015
Laravel 5.1	June 2015
Laravel 5.2	December 2015
Laravel 5.3	August 23, 2016
Laravel 5.4	January 24, 2017
Laravel 6	September 3, 2019
Laravel 7	March 3, 2020
Laravel 8	September 8, 2020
Laravel 9	February 8, 2022

Laravel was developed and created by **Taylor Otwell** as an attempt to give an excellent substitute for the older PHP framework named **CodeIgniter**. And this was because CodeIgniter did not offer such great features as support for built-in customer authentication and proper user authorization. In **June 2011** Laravel released its first beta version, and later in the same month, **Laravel 1** got released. Other than authentication, Laravel also has built-in support for localization, views, dealing with sessions, routing the request to the specific controller, and other amazing features.

Laravel – Short Introduction

Installation

✓ Composer

If you don't have Composer installed on your computer, first visit this URL to download Composer: https://getcomposer.org/download/

✓ Setup Laravel using Installer

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

- Create a Laravel project

```
composer create-project laravel/laravel folder_name --prefer-dist
```

- Start the Laravel service

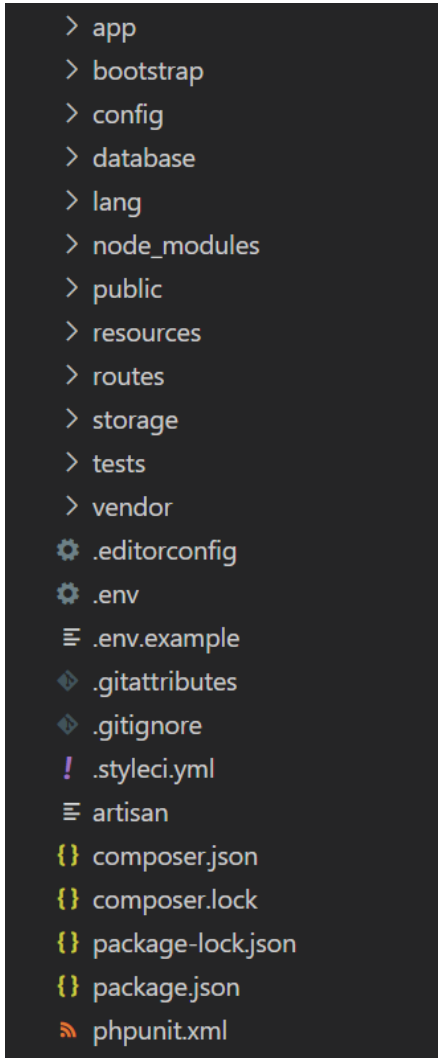
```
php artisan serve
```

Laravel Development server started on **http://localhost:8080**.

Laravel – Short Introduction

- The Root Directory Structure of Laravel

Directory	Description
app	The app directory holds the base code for your Laravel application.
bootstrap	The bootstrap directory contains all the bootstrapping scripts used for your application.
config	The config directory holds all your project configuration files (.config).
database	The database directory contains your database files.
public	The public directory helps start your Laravel project and maintains other necessary files such as JavaScript, CSS, and images of your project.
resources	The resources directory holds all the Sass files, language (localization) files, and templates (if any).
routes	The routes directory contains all your definition files for routing, such as console.php, api.php, channels.php, etc.
storage	The storage directory holds your session files, cache, compiled templates, and miscellaneous files generated by the framework.
test	The test directory holds all your test cases.
vendor	The vendor directory holds all composer dependency files.



Laravel – Short Introduction

- Generate the application key for session securing and encrypted data keys

```
php artisan key:generate
```

- Configuring the Environment variables

you will need the **.env** file in the project's root directory

```
APP_NAME=Laravel
APP_ENV=local
APP_KEY=base64:OHEW8EMm6lS1w70+Ph
APP_DEBUG=true
APP_URL=http://localhost

LOG_CHANNEL=stack
LOG_DEPRECATIONS_CHANNEL=null
LOG_LEVEL=debug
```

- Configuring the Environment variables

```
'mysql' => [
    'driver' => 'mysql',
    'url' => env('DATABASE_URL'),
    'host' => env('DB_HOST', '127.0.0.1'),
    'port' => env('DB_PORT', '3306'),
    'database' => env('DB_DATABASE', 'forge'),
    'username' => env('DB_USERNAME', 'forge'),
    'password' => env('DB_PASSWORD', ''),
    'unix_socket' => env('DB_SOCKET', ''),
    'charset' => 'utf8mb4',
    'collation' => 'utf8mb4_unicode_ci',
```

Laravel – Short Introduction

- Create Routes in Laravel

http://localhost/

```
Route:: get ('/', function () {
   return 'Welcome to index';
});
```

http://localhost/user/dashboard

```
Route:: post('user/dashboard', function () {
   return 'Welcome to dashboard';
});
```

http://localhost/user/add

```
Route:: put('user/add', function () {
//
});
```

http://localhost/post/example

```
Route:: delete('post/example', function () {
//
});
```

Example:

app/Http/routes.php

```
<?php

Route:: get ('/', function () {
   return view('laravel');
});
```

resources/view/laravel.blade.php

```
<html>
<head>
   <title>Laravel5 Tutorial</title>
</head>

<body>
   <h2>Laravel5 Tutorial</h2>
   <p>Welcome to Laravel5 tutorial.</p>
</body>

</html>
```

Laravel – Short Introduction

- Route Middleware

Example:

```
rotected $routeMiddleware = [
    'auth' => \Illuminate\Auth\Middleware\Authenticate::class,
    'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
    'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
    'userAuth' => \Illuminate\Routing\Middleware\UserAuthRequests::class,
];
```

Laravel – Short Introduction

- Controller

Create the controller:

Syntax:

```
php artisan make:controller <controller-name>
```

It can be invoked from within the **routes.php** file using this syntax below-

Example:

```
Route::get('base URI','controller@method');
```

Laravel – Short Introduction

- Controller Middleware

Example:

```
Route::get('profile', 'AdminController@show')->middleware('auth');
```

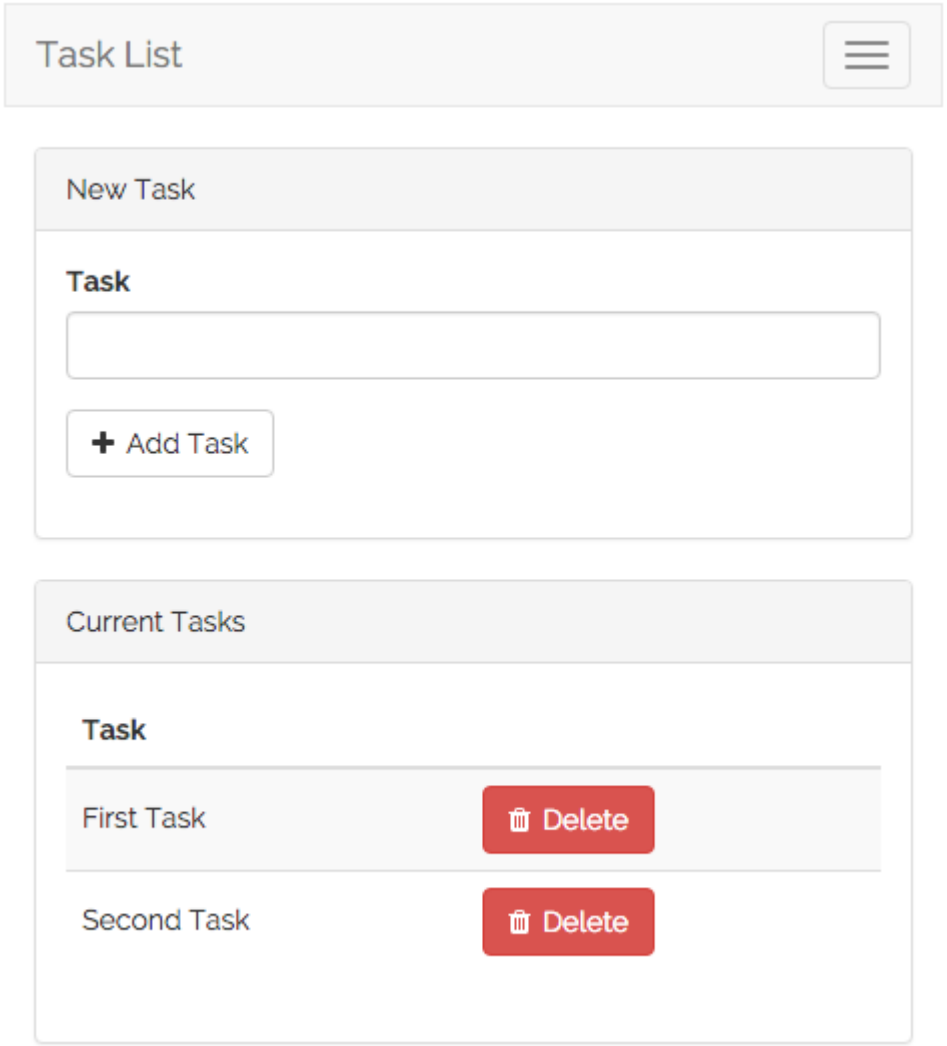
It can also be provided to middleware's on the controller class.

```
class AdminController extends Controller
{
    public function __construct()
    {
        // function body
    }
}
```

Practical exercise

 **Submission :: use the exercise as TP**

Task List


Task List
New Task
Task
Add Task
Current Tasks
Task
First Task
Delete
Second Task
Delete

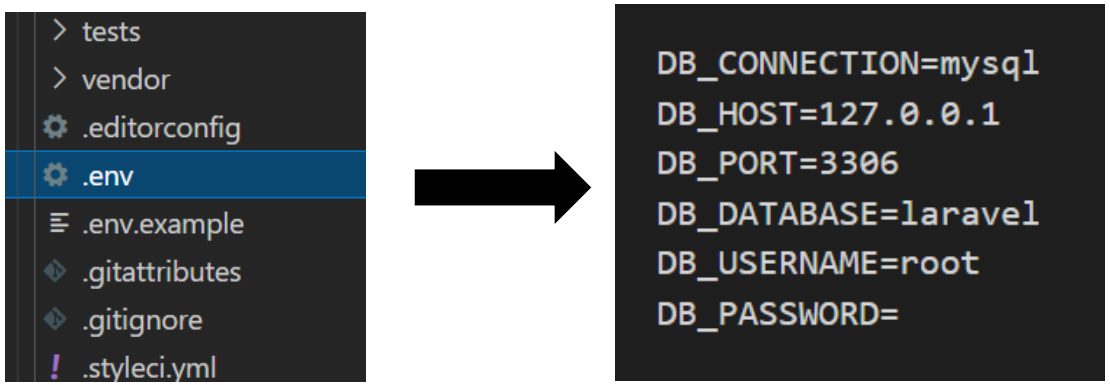
Installation

- Install the Laravel framework using Composer:

```
composer create-project laravel/laravel tasklist --prefer-dist
```

Project name

- Create database and configure the connection (MySQL)



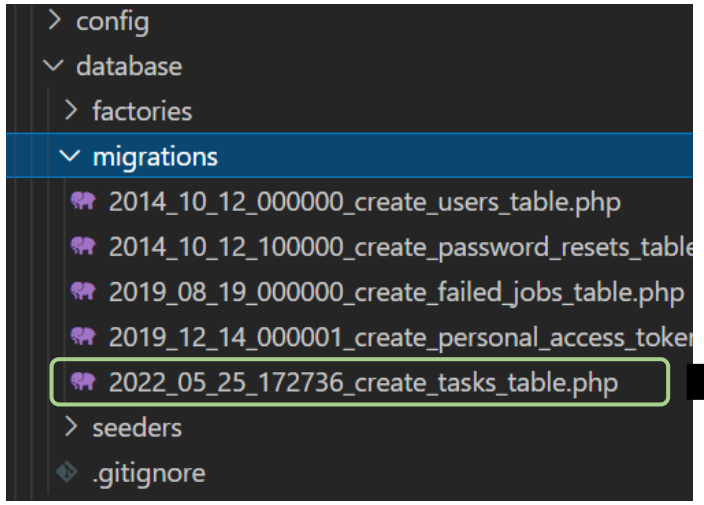
- Install its dependencies

```
cd tasklist
composer install
php artisan migrate
```

Database Migrations

- Let's build a database table that will hold all of our tasks

```
php artisan make:migration create_tasks_table --create=tasks
```



☞ Add an additional string column for the task name

```php
<?php

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

return new class extends Migration
{
    public function up()
    {
        Schema::create('tasks', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::dropIfExists('tasks');
    }
};

```

- Migrate a new task table

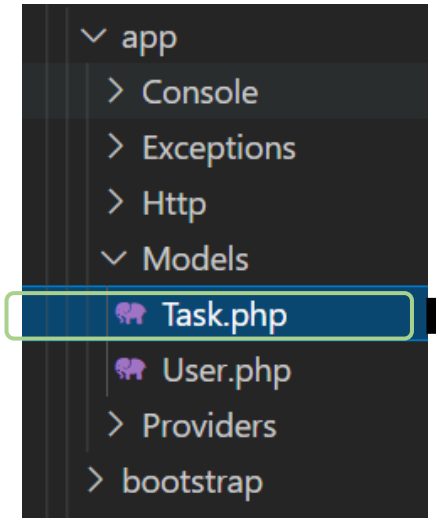
```
php artisan migrate
```

Eloquent Models

Eloquent is Laravel's default ORM (object-relational mapper). Eloquent makes it painless to retrieve and store data in your database using clearly defined "models".

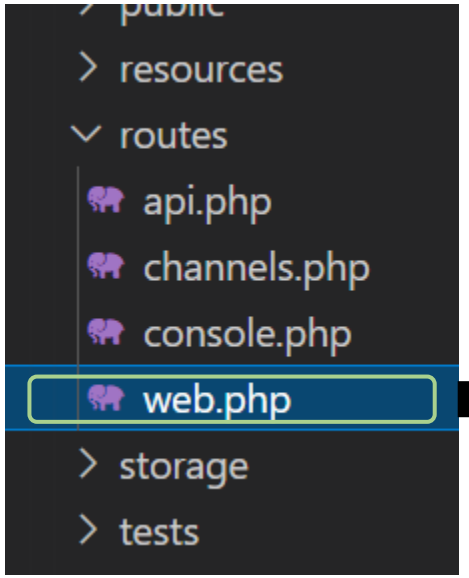
- Define a Task model corresponding to tasks table we just created

```
php artisan make:model Task
```




```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Task extends Model
{
    use HasFactory;
}

```

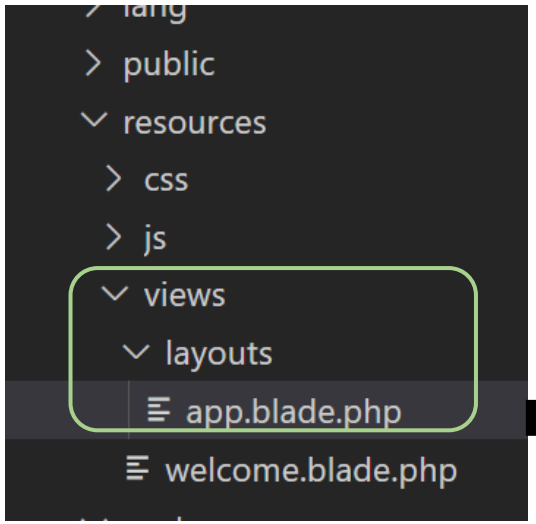
Routing




```php
<?php

use Illuminate\Support\Facades\Route;
use Illuminate\Support\Facades\Request;

Route::get('/', function () {
    return view('welcome');
});

/**
 * Add A New Task
 */
Route::post('/task', function (Request $request) {
    //
});

/**
 * Delete An Existing Task
 */
Route::delete('/task/{id}', function ($id) {
    //
});
```

Displaying A View

- Building Layouts & Views
  - Defining The Layout

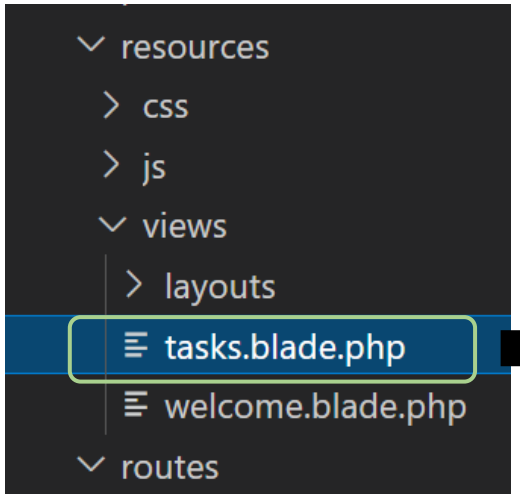



```blade
// resources/views/layouts/app.blade.php

<!DOCTYPE html>
<html lang="en">
    <head>
        <title>Laravel Quickstart - Basic</title>

        <!-- CSS And JavaScript -->
    </head>

    <body>
        <div class="container">
            <nav class="navbar navbar-default">
                <!-- Navbar Contents -->
            </nav>
        </div>

        @yield('content')
    </body>
</html>
```

This is a special Blade directive that specifies where all child pages that extend the layout can inject their own content

Displaying A View

- Building Layouts & Views
  - Defining The Child View



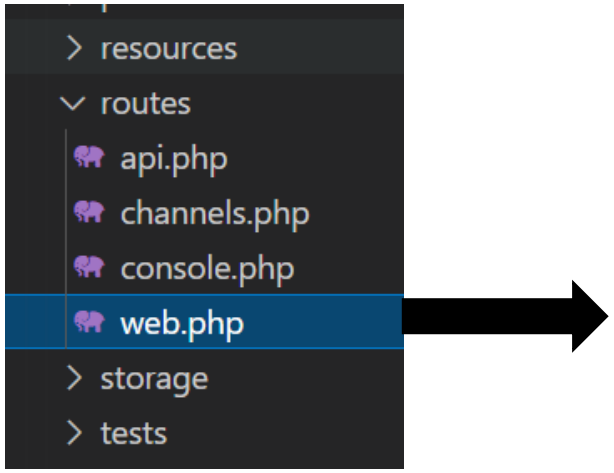

```blade
@extends('layouts.app')

@section('content')
    <!-- Bootstrap Boilerplate... -->

    <div class="panel-body">

        <!-- New Task Form -->
        <form action="/task" method="POST" class="form-horizontal">
            {{ csrf_field() }}

            <!-- Task Name -->
            <div class="form-group">
                <label for="task" class="col-sm-3 control-label">Task</label>

                <div class="col-sm-6">
                    <input type="text" name="name" id="task-name" class="form-control">
                </div>
            </div>

            <!-- Add Task Button -->
            <div class="form-group">
                <div class="col-sm-offset-3 col-sm-6">
                    <button type="submit" class="btn btn-default">
                        <i class="fa fa-plus"></i> Add Task
                    </button>
                </div>
            </div>
        </form>
    </div>

    <!-- TODO: Current Tasks -->
@endsection
```

  - Add a view to route

```php
Route::get('/', function () {
    return view('tasks');
});
```

Adding Tasks

- Creating task with Validation



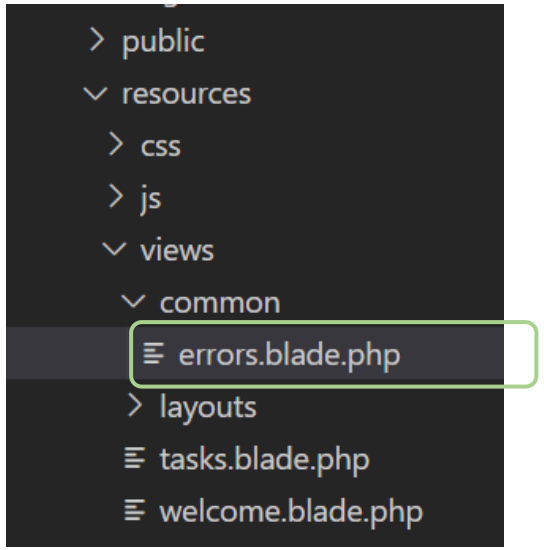

```php
<?php

use App\Models\Task;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;
use Illuminate\Support\Facades\Validator;

Route::get('/', function () {
    return view('tasks');
});

Route::post('/task', function (Request $request) {
    $validator = Validator::make($request->all(), [
        'name' => 'required|max:255',
    ]);

    if ($validator->fails()) {
        return redirect('/')
            ->withInput()
            ->withErrors($validator);
    }

    $task = new Task;
    $task->name = $request->name;
    $task->save();

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

Request validation

Task creation into database

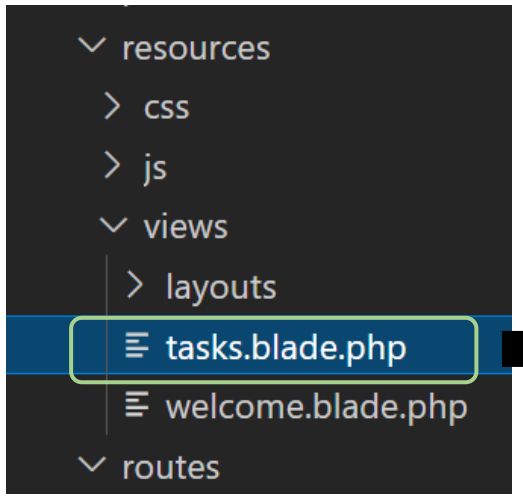
Adding Tasks

- Error handler




```
@if (count($errors) > 0)
    <!-- Form Error List -->
    <div class="alert alert-danger">
        <strong>Whoops! Something went wrong!</strong>

        <br><br>

        <ul>
            @foreach ($errors->all() as $error)
                <li>{{ $error }}</li>
            @endforeach
        </ul>
    </div>
@endif
```

- Include error handler file into task creation form




```
@extends('layouts.app')

@section('content')

    <!-- Bootstrap Boilerplate... -->

    <div class="panel-body">
        <!-- Display Validation Errors -->
        @include('common.errors')

        <!-- New Task Form -->
```

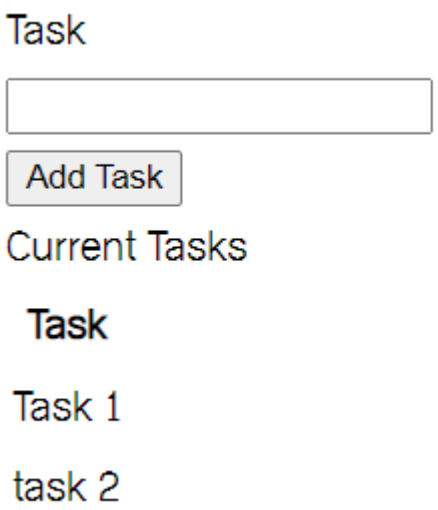
Display Existing tasks

- Displaying Tasks

Pass all of the existing tasks from the database to the view

```php
Route::get('/', function () {
    $tasks = Task::orderBy('created_at', 'asc')->get();

    return view('tasks', [
        'tasks' => $tasks
    ]);
});
```

routes\web.php



resources\views\tasks.blade.php

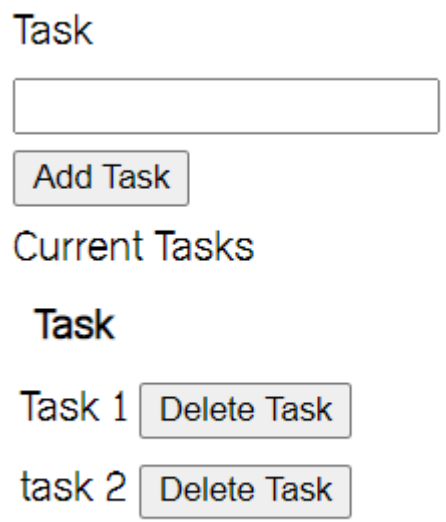
```blade
    <!-- TODO: Current Tasks -->
    <!-- Current Tasks -->
    @if (count($tasks) > 0)
        <div class="panel panel-default">
            <div class="panel-heading">
                Current Tasks
            </div>

            <div class="panel-body">
                <table class="table table-striped task-table">

                    <!-- Table Headings -->
                    <thead>
                        <th>Task</th>
                        <th> </th>
                    </thead>

                    <!-- Table Body -->
                    <tbody>
                        @foreach ($tasks as $task)
                            <tr>
                                <!-- Task Name -->
                                <td class="table-text">
                                    <div>{{ $task->name }}</div>
                                </td>

                                <td>
                                    <!-- TODO: Delete Button -->
                                </td>
                            </tr>
                        @endforeach
                    </tbody>
                </table>
            </div>
        </div>
    @endif
@endsection
```

Deleting Tasks

- Adding The Delete Button

resources\views\tasks.blade.php

```
<tr>
    <!-- Task Name -->
    <td class="table-text">
        <div>{{ $task->name }}</div>
    </td>

    <!-- Delete Button -->
    <td>
        <form action="/task/{{ $task->id }}" method="POST">
            {{ csrf_field() }}
            {{ method_field('DELETE') }}

            <button>Delete Task</button>
        </form>
    </td>
</tr>
```



- let's add logic to our route to actually delete the given task

routes\web.php

```
Route::delete('/task/{id}', function ($id) {
    Task::findOrFail($id)->delete();

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

Task List Exercise Ref:
https://laravel.com/docs/5.1/quickstart#displaying-existing-tasks

Good luck 👌