Web Framework Programming

Topik 12:

Laravel Authentication

WEEK 12

Informatics Engineering

Universitas Surabaya

Authentication

https://laravel.com/docs/10.x/authentication

Authentication

Authentication is a process of proving or validating users who access a website.

- Authentication Purpose
  - Maintain data access security in the system
  - Manage data according to their duties and responsibilities

- Implementation in real applications
  - Login
  - Register
  - Logout

Laravel Authentication

- Authentication in Laravel is quite simple by utilizing the available classes.
- Programmers can configure the application in the config/auth.php file.
- As provided in the image, there is a configuration where Laravel Authentication **uses the users table that has been created through migration.**
- If you haven't run the migration, then run the migration on the users tables with the standard script provided by Laravel.
- Make sure Model Users is available in App/Models.

Authentication Quick Start

1. Preparation
    1. Backup your route/web.php code, because after install, your route/web.php will be replace with route with auth.
    2. Install composer (already installed in the first week)
    3. Install Node.js

2. Install Laravel UI
    1. Install Auth UI Bootstrap Component
    2. Install packages using NPM
    3. Run installed packages

Preparation

1. Install Node.js

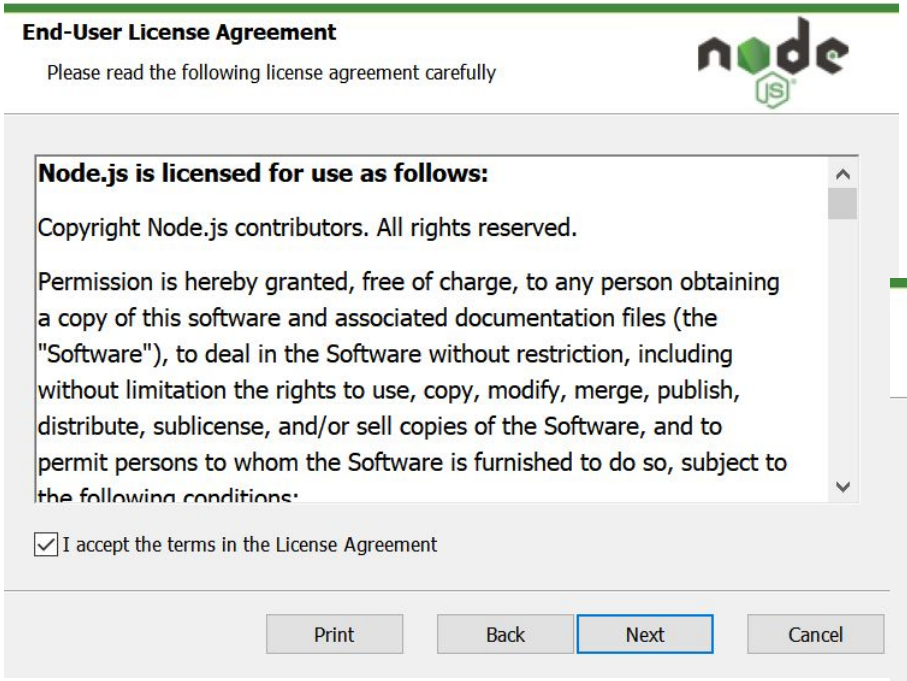
```
PS C:\xampp\htdocs\lara10Auth> npm install
npm : The term 'npm' is not recognized as the name of a cmdlet, function, script file, or operable program. Check the spelling of the
name, or if a path was included, verify that the path is correct and try again.
At line:1 char:1
+ npm install
+ ~~~
    + CategoryInfo          : ObjectNotFound: (npm:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException
```

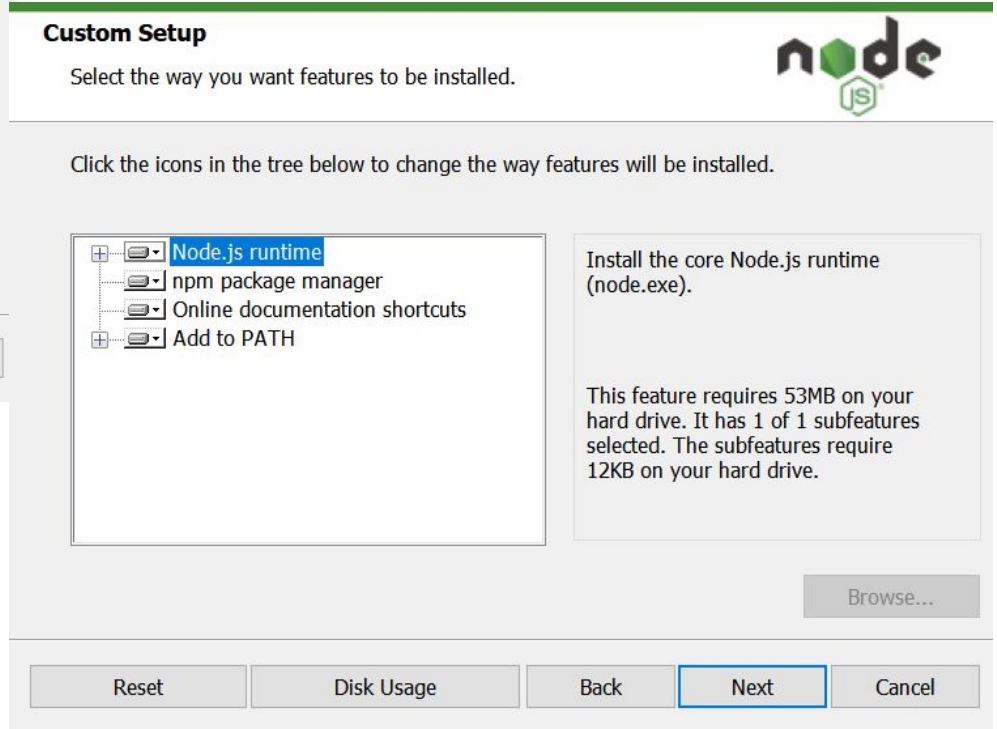
If you haven't installed **Node.js**, the "npm" command in the CLI will not available and an error will appear as in the image above.

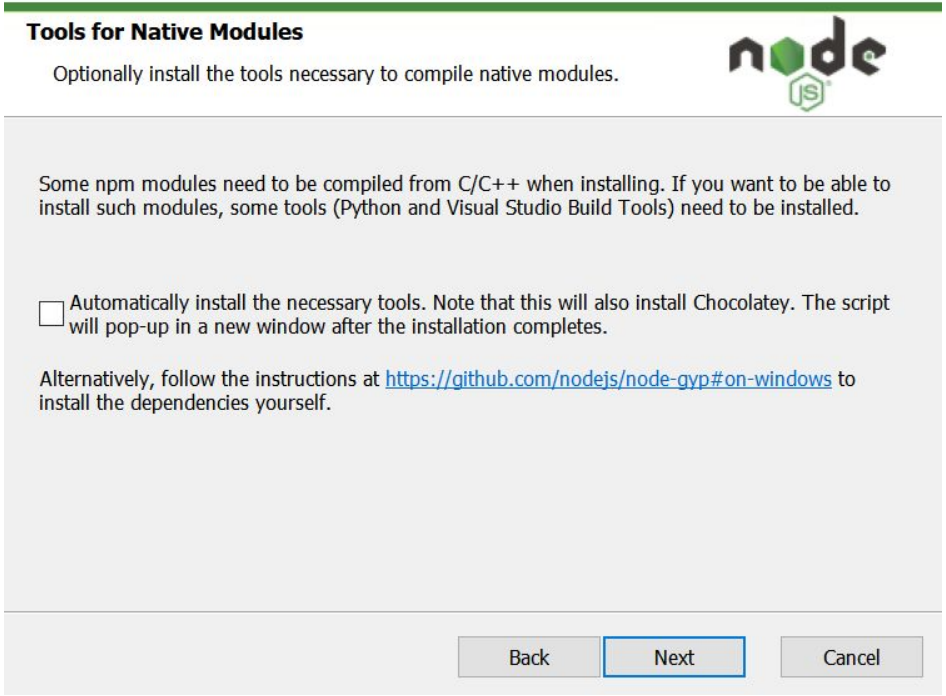
Install Node.js (Approx. 25 MB)

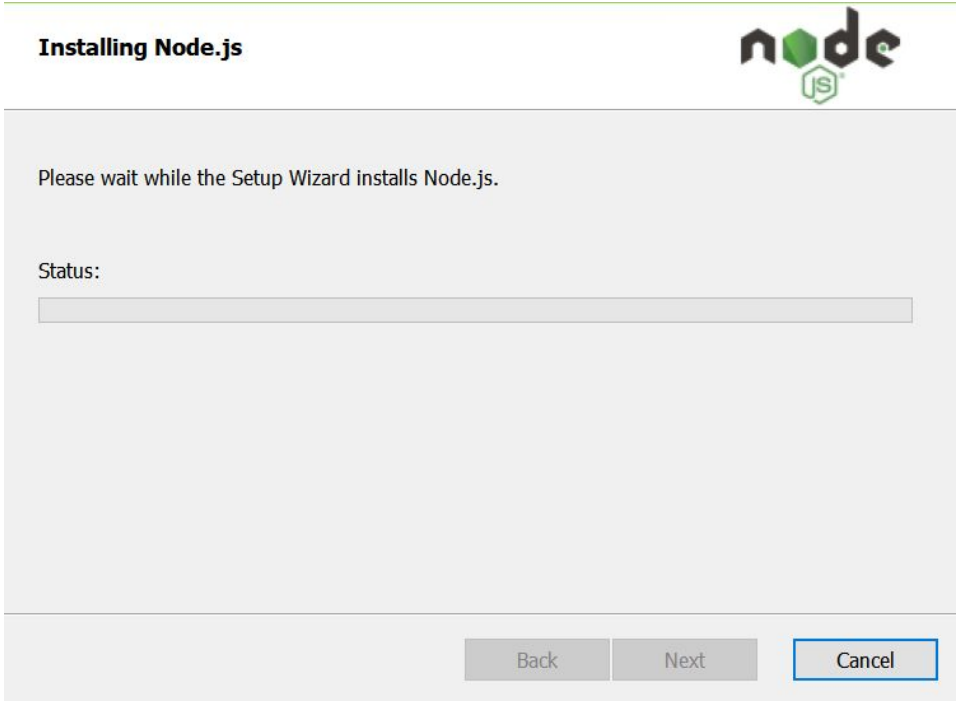
https://nodejs.org/en/download/prebuilt-installer

Download the LTS version !


End-User License Agreement
Please read the following license agreement carefully
Node.js is licensed for use as follows:
Copyright Node.js contributors. All rights reserved.
Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:
I accept the terms in the License Agreement
Print
Back
Next
Cancel


Custom Setup
Select the way you want features to be installed.
Click the icons in the tree below to change the way features will be installed.
Node.js runtime
npm package manager
Online documentation shortcuts
Add to PATH
Install the core Node.js runtime (node.exe).
This feature requires 53MB on your hard drive. It has 1 of 1 subfeatures selected. The subfeatures require 12KB on your hard drive.
Browse...
Reset
Disk Usage
Back
Next
Cancel


Tools for Native Modules
Optionally install the tools necessary to compile native modules.
Some npm modules need to be compiled from C/C++ when installing. If you want to be able to install such modules, some tools (Python and Visual Studio Build Tools) need to be installed.
Automatically install the necessary tools. Note that this will also install Chocolatey. The script will pop-up in a new window after the installation completes.
Alternatively, follow the instructions at https://github.com/nodejs/node-gyp#on-windows to install the dependencies yourself.
Back
Next
Cancel


Installing Node.js
Please wait while the Setup Wizard installs Node.js.
Status:
Back
Next
Cancel

Test NPM Command

Run **npm**

```
PS C:\xampp74\htdocs\wfp\mystore> npm

Usage: npm <command>

where <command> is one of:
    access, adduser, audit, bin, bugs, c, cache, ci, cit,
    clean-install, clean-install-test, completion, config,
    create, ddp, dedupe, deprecate, dist-tag, docs, doctor,
    edit, explore, fund, get, help, help-search, hook, i, init,
    install, install-ci-test, install-test, it, link, list, ln,
    login, logout, ls, org, outdated, owner, pack, ping, prefix,
    profile, prune, publish, rb, rebuild, repo, restart, root,
    run, run-script, s, se, search, set, shrinkwrap, star,
    stars, start, stop, t, team, test, token, tst, un,
    uninstall, unpublish, unstar, up, update, v, version, view,
    whoami
```

2. Install Laravel UI

Run this command in command prompt / terminal

```
composer require laravel/ui
```

```
PS C:\xampp\htdocs\lara10auth> composer require laravel/ui
./composer.json has been updated
Running composer update laravel/ui
Loading composer repositories with package information
Updating dependencies
Lock file operations: 1 install, 0 updates, 0 removals
  - Locking laravel/ui (v4.5.2)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
  - Downloading laravel/ui (v4.5.2)
  - Installing laravel/ui (v4.5.2): Extracting archive
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi
```

Auth UI Vendor Alternatives

Bootstrap

```
php artisan ui bootstrap --auth
```

Vue JS

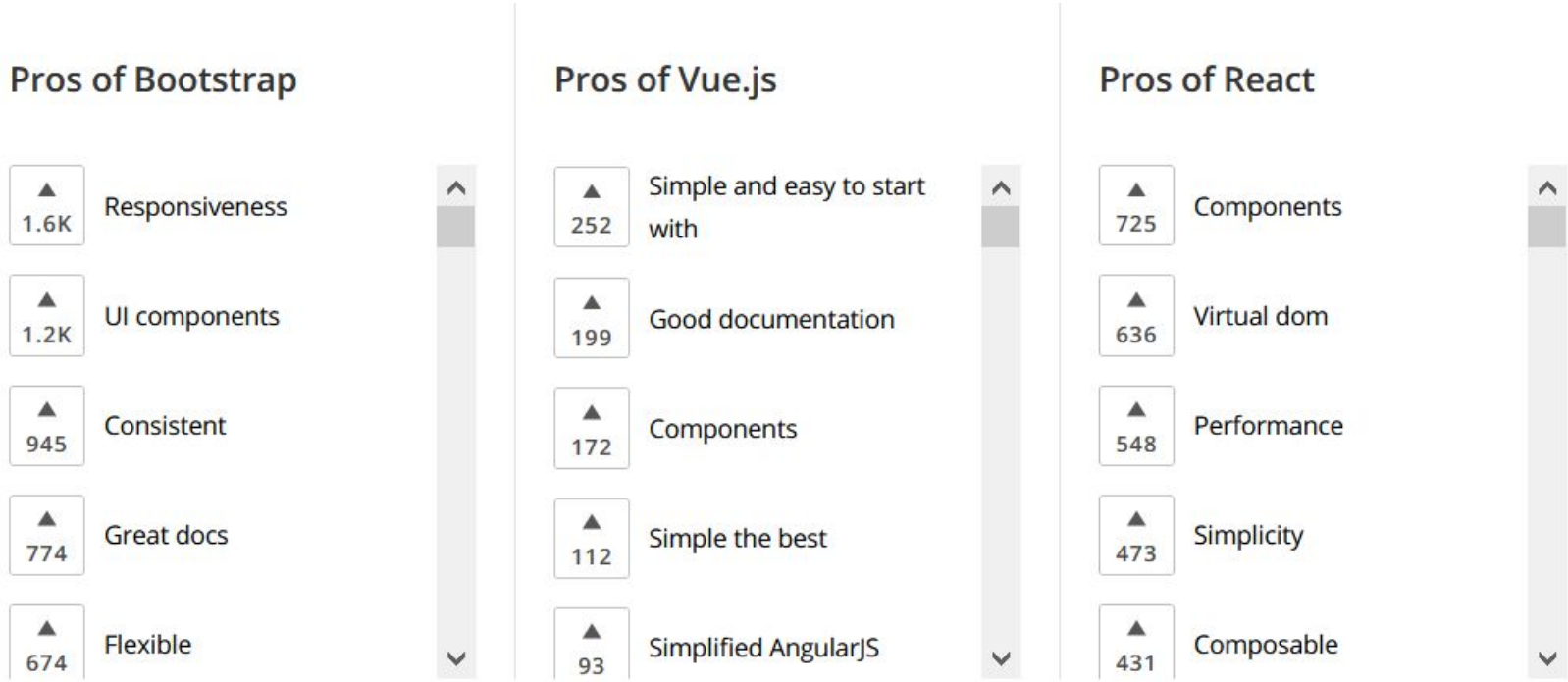
```
php artisan ui vue --auth
```

React JS

```
php artisan ui react --auth
```

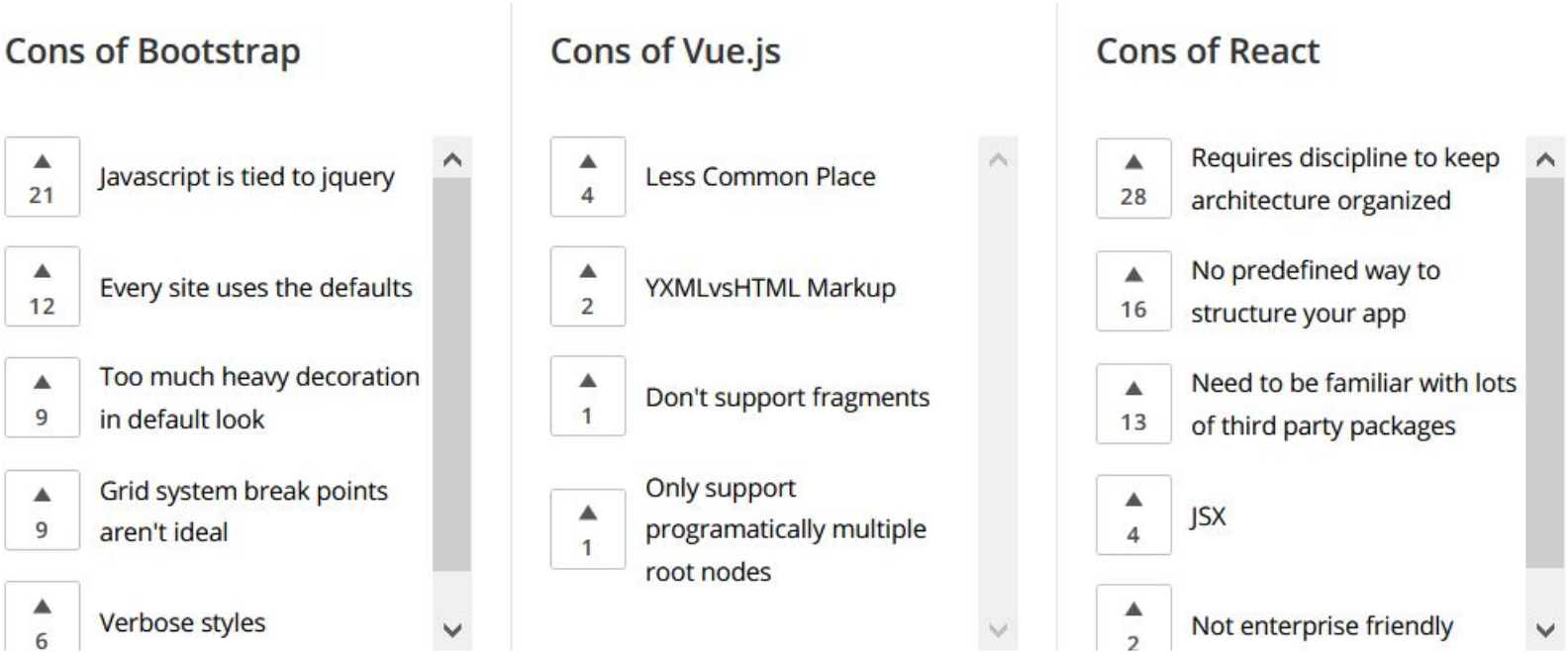
Layout Vendor Comparison(Pros)

https://stackshare.io/stackups/bootstrap-vs-vue-js-vs-react



Layout Vendor Comparison(Cons)

https://stackshare.io/stackups/bootstrap-vs-vue-js-vs-react



2.1 Install Auth UI Bootstrap Component

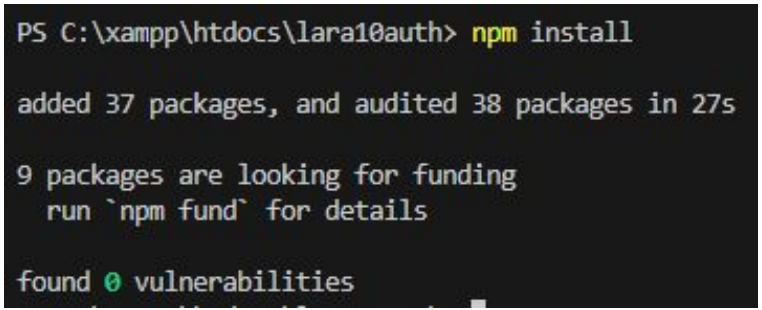
Run this following command using terminal / CLI (Backup App/Http/Controllers/Controller.php first !)

```
php artisan ui bootstrap --auth
```

2.2 Install packages using NPM

Run the following command from terminal / CLI (Warning : You must already install Node.js)

```
npm install
```

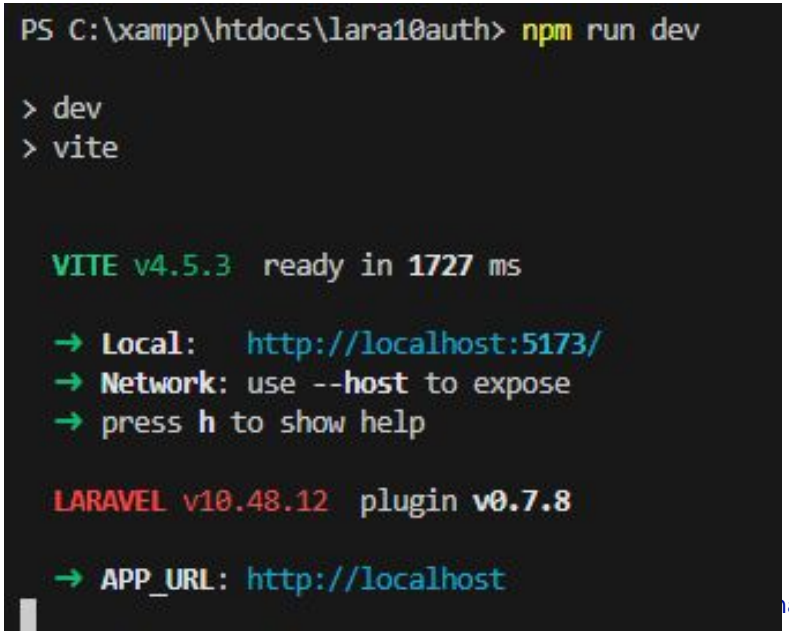


Make sure it found 0 vulnerabilities

2.3 Run installed packages

Run the following command from terminal / CLI (Warning : You must already install Node.js)

```
npm run dev
```



Addition file inside your Laravel

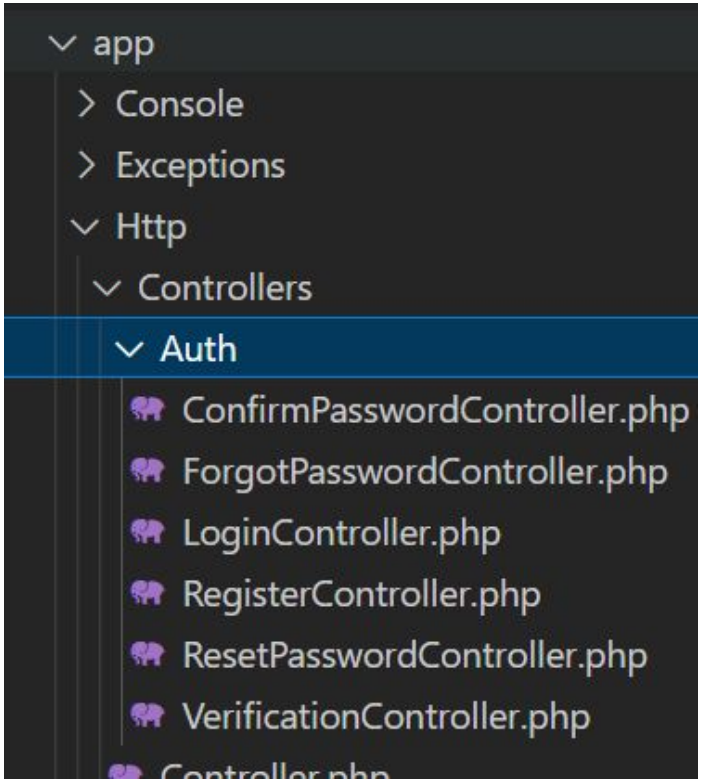
New routes for authentication process

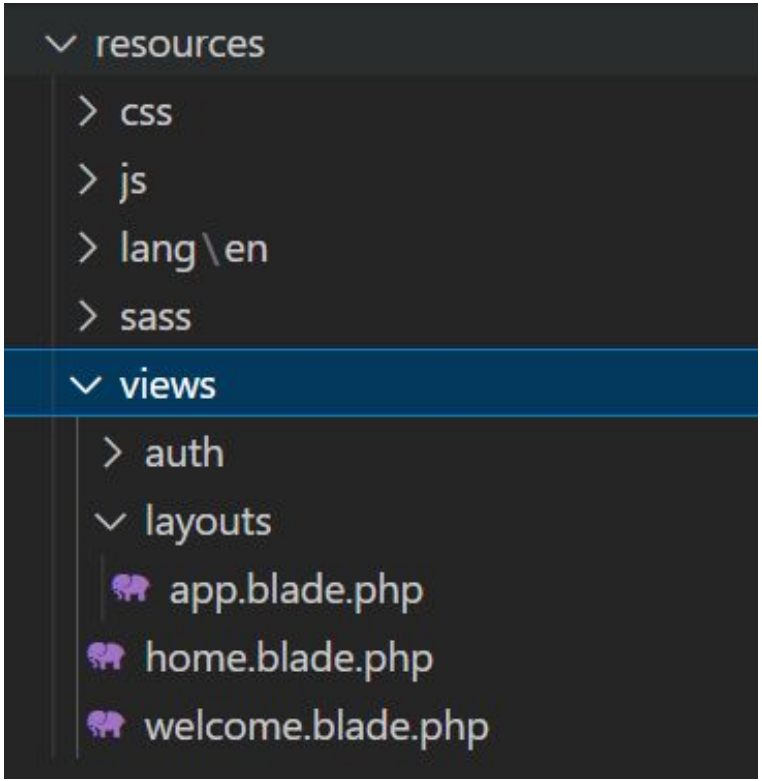
```
Route::get('/', function () {
    return view('welcome');
});

Auth::routes();

Route::get('/home', [App\Http\Controllers\HomeController::class, 'index'])->name('home');
```

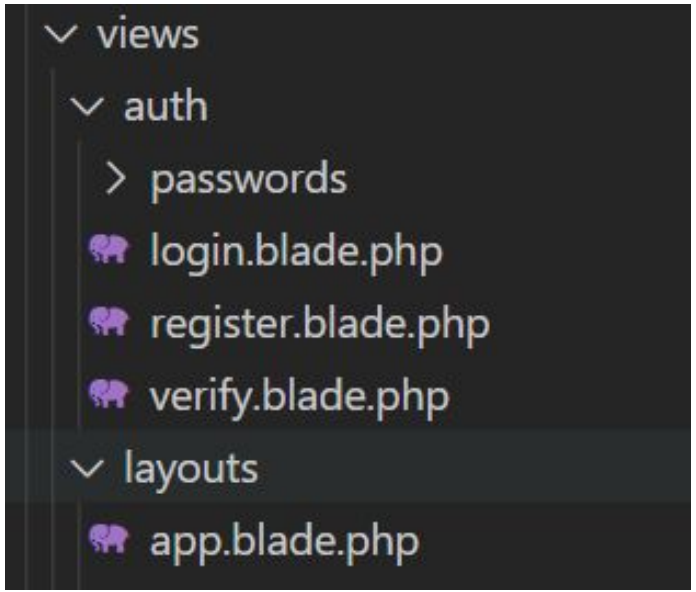
Addition file inside your Laravel





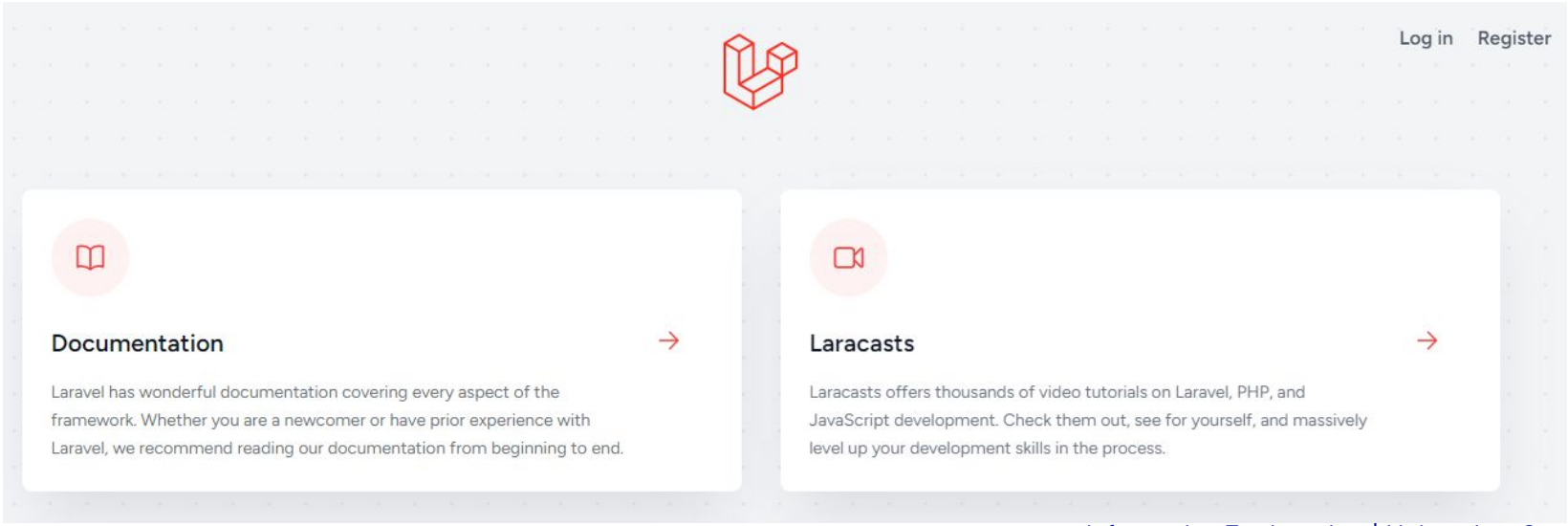
View for Authentication

New layout and view resource in auth and layouts folder in resource/view

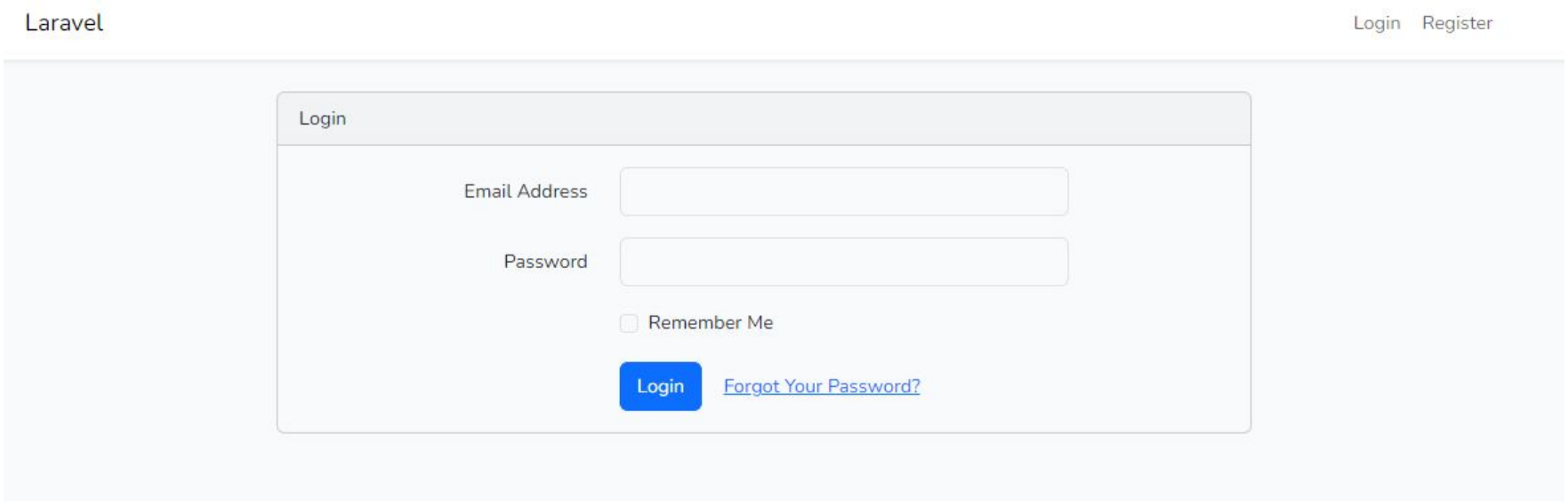


Authentication page

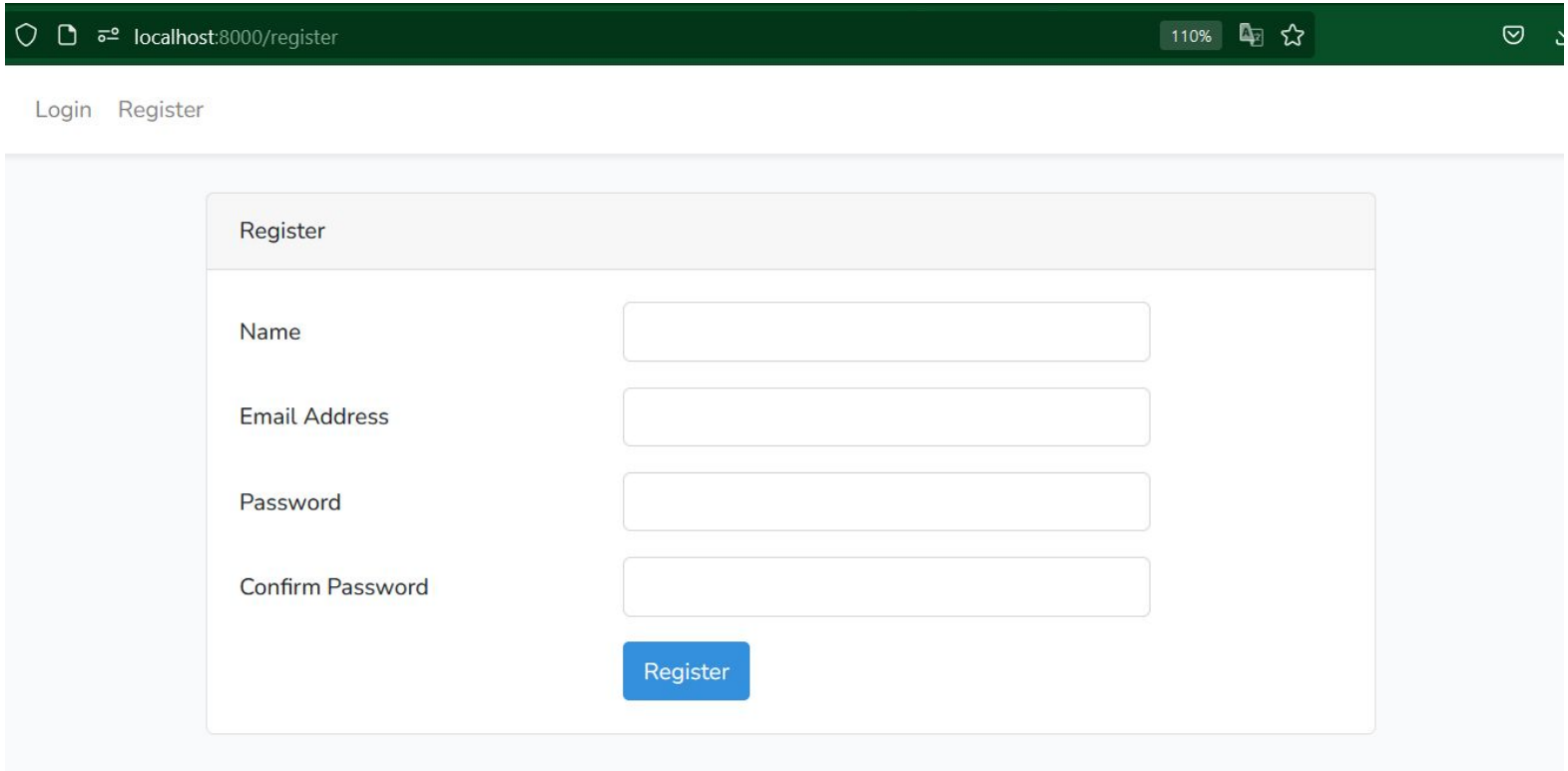
When we access localhost:8000, the User Interface will show Login/Register button



Login Form


Laravel
Login
Register
Login
Email Address
Password
Remember Me
Login
Forgot Your Password?

Register Form



CASE #1: Authentication Usage

CRUD of Type data can be accessed after login

https://laravel.com/docs/10.x/authentication#protecting-routes

At first...

You can access the Citizen data URL freely without requiring a login, even if you have implemented the login above. To limit the Administration Citizen section, we need to modify the routing or controller

Implement Laravel Middleware

https://laravel.com/docs/10.x/middleware

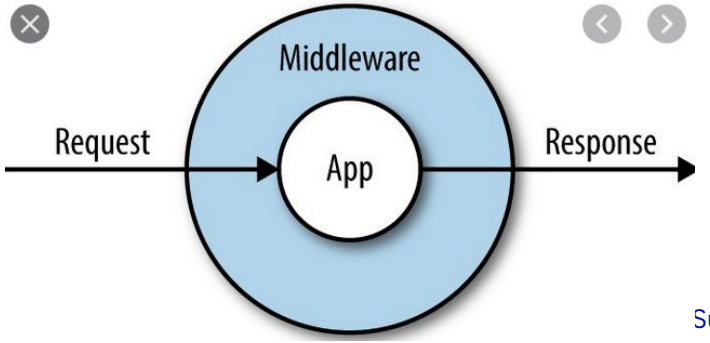
Middleware is a mechanism for filtering all requests that may enter an application
Laravel has a default middleware, namely "auth" where this middleware means that all requests on the routing must be successfully authenticated (must be logged in). The session auth file is located in the following directory with the package class below.

Implementation of the syntax :

vendor > laravel > framework > src > Illuminate > Session > Middleware > AuthenticateSession.php

```
\Illuminate\Session\Middleware\AuthenticateSession::class,
```

```
Route::get('profile', function () {
    // Only authenticated users may enter...
})->middleware('auth');
```



Limit Access to Citizen using Middleware

Look for the URL or scope of the program that processes Citizen data

There are 2 ways of implementation, namely :
give one by one the statements -> middleware('auth')

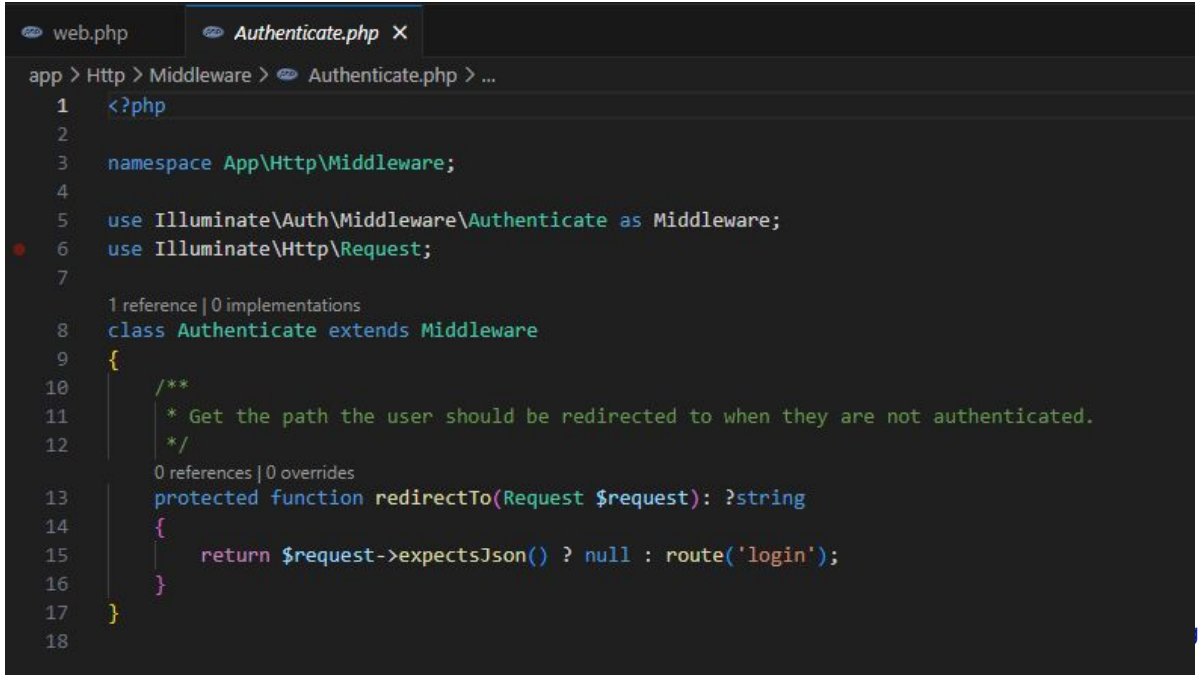
```
Route::resource("/citizen", CitizenController::class)->middleware("auth");
```

Using Middleware Group
(https://laravel.com/docs/10.x/middleware#middleware-groups) by implementing the following syntax:

```
Route::middleware(["auth"])->group(function(){
  Route::resource("/citizen", CitizenController::class);
});
```

Additional Feature Regarding Authentication

You can change the redirect if it is not authenticated by the middleware ('auth'). You can access it in App\Http\Middleware. Select the file that corresponds to the middleware. If 'auth', then access Authenticate.php



Additional Feature Regarding Authentication

2. You can retrieve the session data of who is logged in by using Auth class and you display it in the view.

```
i class="nav-item dropdown">
  <a id="navbarDropdown" class="nav-link dropdown-toggle"
      {{ Auth::user()->name }}
  </a>
```

3. The Auth class can also be used on certain controllers or classes

```php
use Illuminate\Support\Facades\Auth;

// Get the currently authenticated user...
$user = Auth::user();

// Get the currently authenticated user's ID...
$id = Auth::id();
```

Exercise

Implement authentication with bootstrap and limit access to products, contributions, and employee data so that they can only be accessed after logged in.

Authorization

https://laravel.com/docs/10.x/authorization

Authorization

Authorization is a function for specify access and privileges for a data in the Website.
another terminology : policy
Why use Authorization

- Maintain security of data in the system
- Ensure that users who view and manage data are in accordance with their duties and responsibilities

Authorization in Laravel

Authorization in Laravel using GATES and POLICY class

Gates is a “clause” in laravel which determine whether user is allowed to do something
Located on App\Providers\AuthServiceProvider class

- Policies is a class which manage the logic of athorization in modal or another resource
- Located on app\Policies

```
public function boot()
{
    $this->registerPolicies();

    Gate::define('update-post', 'App\Policies\PostPolicy@update');
}
```

```
php artisan make:policy PostPolicy
```

GATES structure

The logic in making GATES is similar with making the ROUTES. Authorization Logic can be used 2 ways:

- write down directly in GATES

```
public function boot()
{
    $this->registerPolicies();

    Gate::define('edit-settings', function ($user) {
        return $user->isAdmin;
    });

    Gate::define('update-post', function ($user, $post) {
        return $user->id === $post->user_id;
    });
}
```

- write down in another controller, the policy

In GATES

```
 */
public function boot()
{
    $this->registerPolicies();

    Gate::define('update-post', 'App\Policies\PostPolicy@update');
}
```

In PostPolicy

```
public function update(User $user, Post $post)
{
    return $user->id === $post->user_id;
}
```

Return/Response of GATES

GATES using Response class.
There are 2 function allow() and deny() along with a related information

```
use Illuminate\Auth\Access\Response;
use Illuminate\Support\Facades\Gate;

Gate::define('edit-settings', function ($user) {
    return $user->isAdmin
                ? Response::allow()
                : Response::deny('You must be a super administrator.');
});
```

Before & After Action in GATES

Gates has before & after function, use for checking the beginning and the end of an authorization handling

```
Gate::before(function ($user, $ability) {
    if ($user->isSuperAdmin()) {
        return true;
    }
});
```

```
Gate::after(function ($user, $ability, $result, $arguments) {
    if ($user->isSuperAdmin()) {
        return true;
    }
});
```

POLICY structure

Policy has a logic similar with Controller. It interact with the model and a function with a TRUE/FALSE .

Matematical expression and logic operation can implement here.

Eloquent model and DB Query also allowed

```php
namespace App\Policies;

use App\Post;
use App\User;

class PostPolicy
{
    /**
     * Determine if the given post can be updated by the user.
     *
     * @param  \App\User  $user
     * @param  \App\Post  $post
     * @return bool
     */
    public function update(User $user, Post $post)
    {
        return $user->id === $post->user_id;
    }
}
```

Return/Response of POLICY

Similar with the GATES, POLICY use Response class.
Also have 2 function allow() and deny() along with the description

```php
use Illuminate\Auth\Access\Response;

/**
 * Determine if the given post can be updated by the user.
 *
 * @param  \App\User  $user
 * @param  \App\Post  $post
 * @return \Illuminate\Auth\Access\Response
 */
public function update(User $user, Post $post)
{
    return $user->id === $post->user_id
                ? Response::allow()
                : Response::deny('You do not own this post.');
}
```

Implementing GATES & POLICY

https://laravel.com/docs/10.x/authorization#authorizing-actions-using-policies

There are 4 ways of implementing GATES & POLICY

1. Via User Model

```php
if ($user->can('update', $post)) {
    //
}
```

2. Via Middleware

```php
use App\Post;

Route::put('/post/{post}', function (Post $post) {
    // The current user may update the post...
})->middleware('can:update,post');
```

Implementing GATES & POLICY

https://laravel.com/docs/10.x/authorization#authorizing-actions-using-policies

3. Via Controller

```php
public function update(Request $request, Post $post)
{
    $this->authorize('update', $post);

    // The current user can update the blog post...
}
```

4. Via View (Blade Template)

```
@can('update', $post)
    <!-- The Current User Can Update The Post -->
@elsecan('create', App\Post::class)
    <!-- The Current User Can Create New Post -->
@endcan

@cannot('update', $post)
    <!-- The Current User Cannot Update The Post -->
@elsecannot('create', App\Post::class)
    <!-- The Current User Cannot Create A New Post -->
@endcannot
```

Pre Exercise – Modify Database

Modify users table

Add new field ‘role’ VARCHAR 50
This field will be used to manage roles. To simplify, we will just 2 roles : ‘manager’ and ‘employee‘

Modify $fillable in Model Eloquent User

```
protected $fillable = [
'name',
'email',
'password',
'role'
];
```

So that the new field can be filled the form

Modify Register Form

Add 1 column form to manage roles of the new user

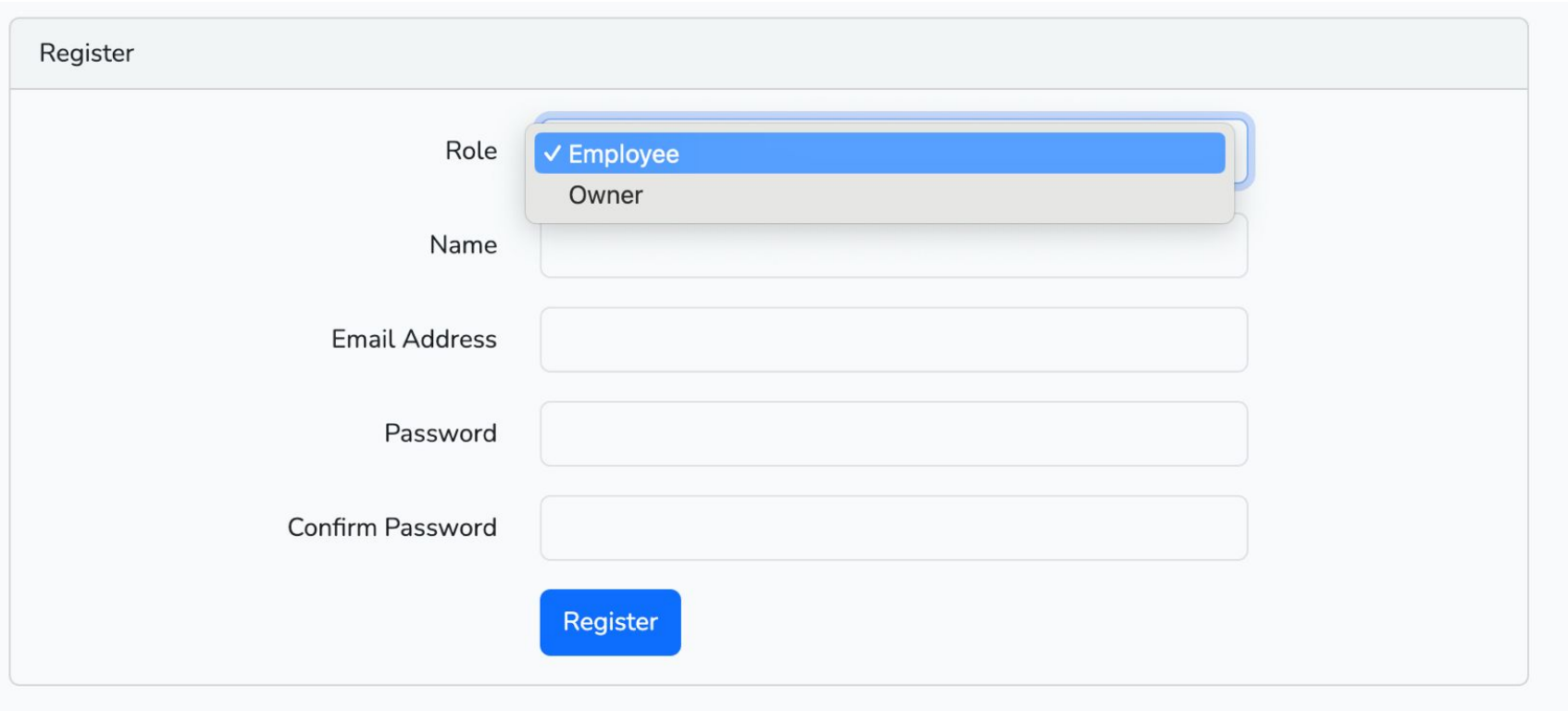
In resources/views/auth/register.blade.php, Add select for UI to select the user role

```
<div class="row mb-3">
  <label for="role" class="col-md-4 col-form-label text-md-end">Role</label>
  <div class="col-md-6">
    <select id="role" type="text" class="form-control" name="role">
      <option value="employee">Employee</option>
      <option value="manager">Manager</option>
    </select>
  </div>
</div>
```

Modify also the controller so it save role into database
open app\Http\Controllers\Auth\RegisterController.php

```php
protected function create(array $data)
{
  return User::create([
    'name' => $data['name’],
    'email' => $data['email’],
    'password' => Hash::make($data['password']),
    'role' => $data['role’]
  ]);
}
```

Result


Register
Role
Employee
Owner
Name
Email Address
Password
Confirm Password
Register

Add 2 user using register form

admin@gmail.com with password admin123 as manager
pegawai@gmail.com with password pegawai123 as employee

See result in Database

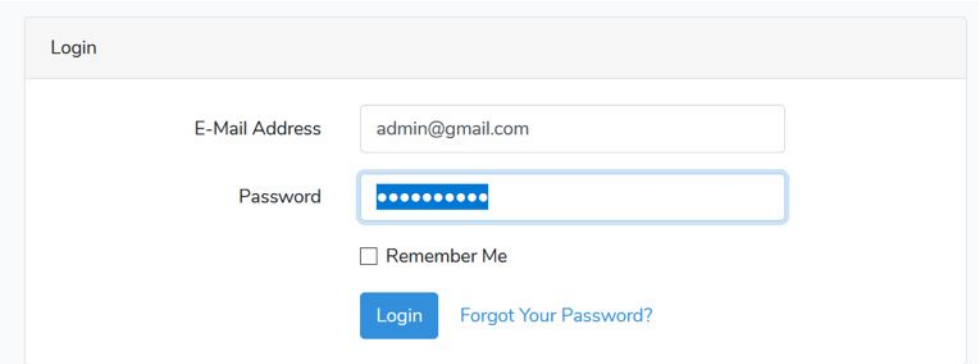
id	name	email	role	password
1	admin	admin@gmail.com	manager	$2y$12$hDMrzeKAiIpntGdamFubuONU9IjVlNuZvA8V/VDEpq4...
2	pegawai	pegawai@gmail.com	employee	$2y$12$HdeF2OhUeNnmkzLoJLrV.em2/vMbq92XBuRTImuEUmI...

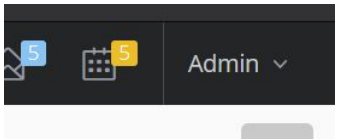
To simulate the authorization, we will redirect after login and register to our previous master citizen

change app\Http\Controller\Auth\LoginController.php parameter $redirectTo = ‘/citizen’
Ubah pada app\Http\Controller\Auth\RegisterController.php parameter $redirectTo = ‘/ citizen’

Try it. You can also try to display the currently logged in user's name

```
<span class="username username-hide-on-mobile">{{ Auth::user()->name }} </span>
```





To Try another user, you need to logout. Add logout in the website

```
<form action="{{ route('logout') }}" method="post">
@csrf<input type="submit" value="logout" class='btn btn-danger' />
</form>
```

Scenario #1: Just Owner can delete Type data

First, define GATES
open app\Providers\AuthServiceProviders.php

2 ways: GATES only

```
public function boot(): void
{
  $this->registerPolicies();
  Gate::define("delete-permisson", function ($user) {
    return ($user->role == "manager") ?
      Response::allow() :
      Response::deny("Only managers are allowed to perform this operation");
  });
}
```

or Gates with Policy

We will use
Gates with Policy

```
public function boot(): void
{
  $this->registerPolicies();
  Gate::define('delete-permission’,
    'App\Policies\CitizenPolicy@delete');
}
```

If we use Gates with Policy, we will need to create new policy file

In CLI, run this command:

```
php artisan make:policy CitizenPolicy
```

It will create new file app\Policy\CitizenPolicy.php

Add delete() function as the definition in GATES

Add this function. It have 1 parameter $user, that will be use to check the Role of the user.

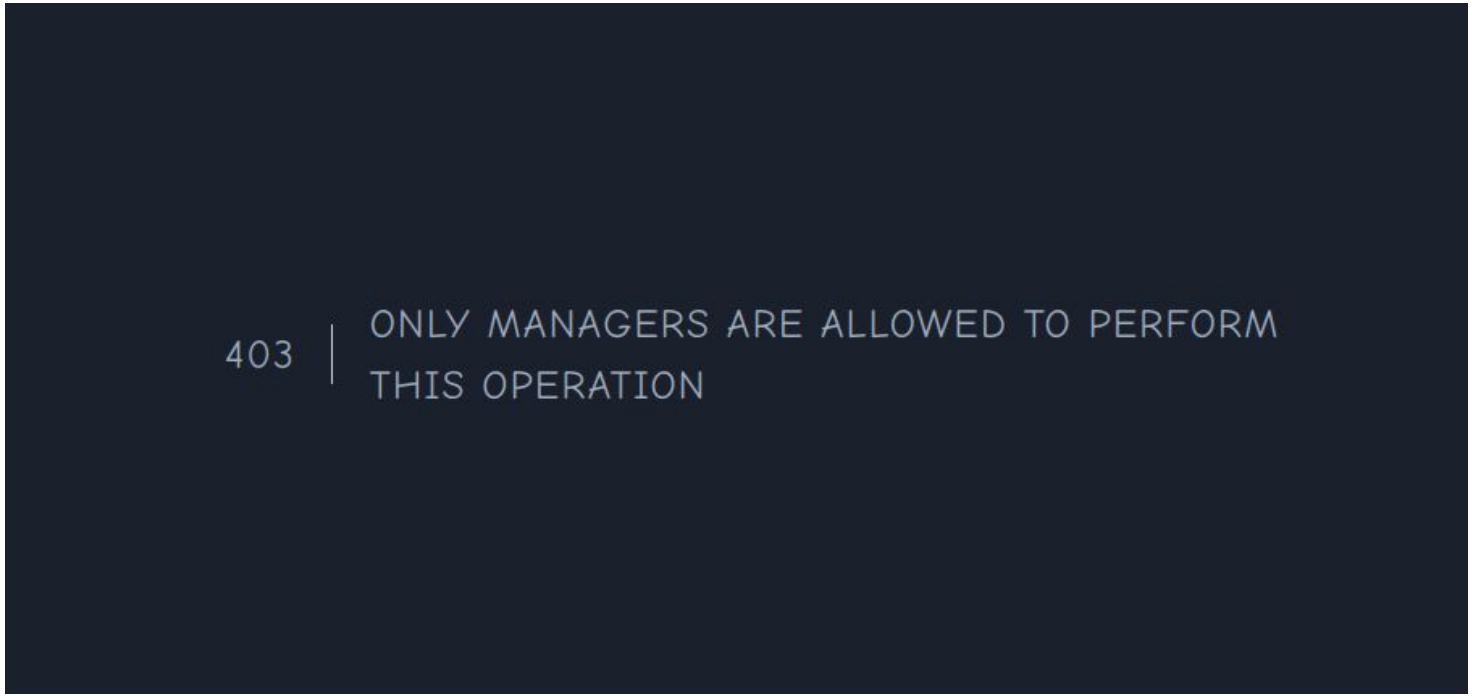
```
public function delete(User $user){
  return ($user->role == "manager") ?
    Response::allow() :
    Response::deny("Only managers are allowed to perform this operation");
}
```

Use GATES & POLICY in Controller when user calling destroy in CitizenController

```php
public function destroy(Citizen $citizen){
  $this->authorize('delete-permission’, Auth::user());
  // ...
}
```

All access to method destroy will firstly facing up the Policy

when you delete with user pegawai :



Scenario #2: Hide DELETE button if user have no authorization

Remember that we have 2 delete buttons in master type screen. We will implement it to our 2nd delete button

Wrap Button or Link (<a>) with @can ....@endcan

```
@can('delete-permission', Auth::user())
  <a href="#" value="DeleteNoReload" class="btn btn-danger btn-xs"
    onclick="if(confirm('Are you sure to delete {{ $r->id }} - {{ $r->name }} ? '))
    deleteDataRemoveTR({{ $r->id }})">
      Delete without Reload
  </a>
@endcan
```

Result

admin

Name	Address	Action
Charlie Davis	789 Oak St, City C	Edit Edit Type A Edit Type B Delete Delete without Reload
Ethan Brown	202 Pine St, City E	Edit Edit Type A Edit Type B Delete Delete without Reload
Bob Smith	456 Elm St, City B	Edit Edit Type A Edit Type B Delete Delete without Reload
Alice Johnson	123 Main St, City A	Edit Edit Type A Edit Type B Delete Delete without Reload
Diana White	101 Maple St, City D	Edit Edit Type A Edit Type B Delete Delete without Reload

pegawai

Name	Address	Action
Charlie Davis	789 Oak St, City C	Edit Edit Type A Edit Type B Delete
Ethan Brown	202 Pine St, City E	Edit Edit Type A Edit Type B Delete
Bob Smith	456 Elm St, City B	Edit Edit Type A Edit Type B Delete
Alice Johnson	123 Main St, City A	Edit Edit Type A Edit Type B Delete
Diana White	101 Maple St, City D	Edit Edit Type A Edit Type B Delete

Exercise

Implement Authentication and Authorization in your Final Project

Thank You