Issue 9 del 29 de enero

Temas

How to code Login & logout

-authorization basics

-route grouping explained

How to code login and logout (1).

Ahora haremos que nuestros usuarios puedan iniciar sesión o salir de ella en nuestra pagina

Comenzamos con la ruta

```
oute::post('/login', [AuthController::class , 'authenticate'] );
```

Añadimos la validación de las casillas email y password
(1)pedimos validar que email y password esten presentes en los campos

(2) le decimos que si los valores que nos dieron son iguales a esos dos mismos tipos de valores en nuestra base de datos

(3)si están ahí y coinciden rediriges al usuario a la pagina principal y si no esta correcto uno de ellos o los dos lo mandas a error con un mensaje personalizado.

```
public function authenticate()
{
    $validated = request()->validate(
        [
            'email' => 'required|email',
            'password' => 'required|confirmed|min:8'
        ]
    );


    if (auth()->attempt($validated)) {
        request()->session()->regenerate();

        return redirect()->route('dashboard')->with('success', 'Logged in successfully!');
    }

    return redirect()->route('login')->withErrors([
        'email' => "No matching user found with the provided email and password"
    ]);
}
}
```

Ahora hagamos que una vez registrado no pueda hacerlo de nuevo con guest(no logeado) y auth(si logeado) los cuales verificaran si el Usuario se ha logeado o no eliminando las etiquetas de register y login

```
@guest
    <li class="nav-item">
        <a class="nav-link active" aria-current="page" href="/login">Login</a>
    </li>
    <li class="nav-item">
        <a class="nav-link" href="/register">Register</a>
    </li>
@endguest
@auth()
    <li class="nav-item">
        <a class="nav-link" href="/profile">Profile</a>
    </li>
@endauth
```

How to code login and logout (2).

Ahora definamos el logout

Creamos su ruta

```
Route::post('/logout', [AuthController::class , 'logout'] )->name('logout');
```

Su respectivo metodo el cual eliminara los datos actuales creara un “token” para redirigir al usuario a la pagina principal

```
public function logout(){
    auth()->logout();

    request()->session()->invalidate();
    request()->session()->regenerateToken();

    return redirect()->route('dashboard')->with('success', 'logged out successfully');
}
```

Y como no su respectivo boton para poder salir de la cuenta

```
<form action="{{ route('logout') }}" method="POST">
    @csrf
    <button class="btn btn-danger btn-sm" type="submit"> Logout </button>
</form>
```

Issue 9 del 29 de enero

authorization basics.

Haremos un ejercicio de autorización en este caso autorizar de eliminar los mensajes solo a los usuarios propietarios

Primero pedimos el id del usuario actual

```
$validated['user_id'] = auth()->id();
```

Tambien debemos añadir la id del usuario actual a lo que se guarda al crear una publicación o un comentario

```
$comment->user_id = auth()->id();
```

También es importante darle la opción a la pagina de que hacer si el usuario no esta registrado y es un invitado ya que de ser asi su id tendrá valor nulo dando erro en vez de eso mejor le decimos que inicie sesión para poder comentar

```
@guest()
    <h4>Login Share yours ideas </h4>
@endguest
```

Una vez con las relaciones hecha podemos asignar el nombre del usuario actual en vez de el de Mario que era nuestro default

```
    src="https://api.dicebear.com/6.x/fun-emoji/svg?seed={{ $idea->user->name }}" alt="{{ $idea->user->name }}">
<div>
    <h5 class="card-title mb-0"><a href="#"> {{ $idea->user->name }}
```

Es necesario añadir esto a las rutas que pidan usuario , es una confirmación de que el usuario este loggeado

```
->middleware('auth');
```

Y para terminar le añadimos un if el cual se asegurara que el usuario loggeado actual es el mismo que creo la publicación o comentario , en este caso si es diferente lo mandamos error y si si es el usuario dejamos que la acción pase

```
if(auth()->id() !== $idea->user_id){
    abort(404);
}
```

Issue 9 del 29 de enero

route grouping explained.

Ahora veremos como crear grupos de ruta los cuales nos servirán para tener un código mas estructurado manteniendo las rutas con mas similitudes en un grupo

Existen 2 formas caso(1) creando la ruta de grupo y las similitudes que tienen ya sea a los métodos que accederán o si tienen middleware o no y la segunda caso(2) es creando un archivo especial para esas rutas en especifico

Caso(1) Antes:

```php
Route::get('/', [DashboardController::class , 'index'] )->name('dashboard');

Route::post('/ideas', [IdeaController::class , 'store'] )->name('ideas.store');

Route::get('/ideas/{idea}', [IdeaController::class , 'show'] )->name('ideas.show');

Route::get('/ideas/{idea}/edit', [IdeaController::class , 'edit'] )->name('ideas.edit')->middleware('auth');

Route::put('/ideas/{idea}', [IdeaController::class , 'update'] )->name('ideas.update')->middleware('auth');;

Route::delete('/ideas/{idea}', [IdeaController::class , 'destroy'] )->name('ideas.destroy')->middleware('auth');;

Route::post('/ideas/{idea}/comments', [CommentController::class , 'store'] )->name('ideas.comments.store')->middleware('
```

Caso(1) Después:

```php
Route::get('', [DashboardController::class, 'index'])->name('dashboard');

Route::group(['prefix' => 'ideas/', 'as' => 'ideas.'], function () {

    Route::post('', [IdeaController::class, 'store'])->name('store');

    Route::get('/{idea}', [IdeaController::class, 'show'])->name('show');

    Route::group(['middleware' => ['auth']], function () {

        Route::get('/{idea}/edit', [IdeaController::class, 'edit'])->name('edit');

        Route::put('/{idea}', [IdeaController::class, 'update'])->name('update');

        Route::delete('/{idea}', [IdeaController::class, 'destroy'])->name('destroy');

        Route::post('/{idea}/comments', [CommentController::class, 'store'])->name('comments.store');
    });
});
```

Caso(2)tenemos que declarar las rutas y de donde se extraeran incluyendo al final la ruta de auth.php que es el archivo especial en el que guardaremos las rutas

```php
public function boot(): void
{
    RateLimiter::for('api', function (Request $request) {
        return Limit::perMinute(60)->by($request->user()?->id ?: $request->ip());
    });

    $this->routes(function () {
        Route::middleware('api')
            ->prefix('api')
            ->group(base_path('routes/api.php'));

        Route::middleware('web')
            ->group(base_path('routes/web.php'));

        Route::middleware('web')
            ->group(base_path('routes/auth.php'));
```

Asi seria como queda el archivo especial de las rutas

```php
use App\Http\Controllers\AuthController;
use Illuminate\Support\Facades\Route;

Route::get('/register', [AuthController::class, 'register'])->name('register');

Route::post('/register', [AuthController::class, 'store']);

Route::get('/login', [AuthController::class, 'login'])->name('login');

Route::post('/login', [AuthController::class, 'authenticate']);

Route::post('/logout', [AuthController::class, 'logout'])->name('logout');
```

Caso(2)además de asegurarnos de poner bien la ruta en nuestro archivo especial

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