CREAZIONE REVISOR

- Creazione is_revisor
Vogliamo creare un altro tipo di utente (oltre all'*utente registrato* e *non registrato*) che possa revisionare gli articoli e decidere se possono essere postati sul sito.

Creiamo la Migration:

```
php artisan make:migration add_is_revisor_flag_to_users
```

Creiamo la colonna *is_revisor* con un ***boolean***.

```
public function up(){
	Schema::table('users', function (Blueprint $table){
		$table->boolean('is_revisor')->default(false);
		});
}

public function down(){
	Schema::table('users', function (Blueprint $table){
		$table->dropColumn('is_revisor');
	});
}
```

Lanciamo la migrazione *php artisan migrate*.

- Creazione Comand
Dobbiamo ***creare un nuovo comando Artisan*** per rendere revisore un determinato utente.

```
php artisan make:comand MakeUserRevisor
```

In *app\Console|Comands:*

```
class MakeUserRevisor extends Command{
	protected $signature = 'presto:makeUserRevisor';
	protected $description = 'Rendi un utente revisore';

	public function __construct(){
		parent::__construct();
	}

	public function handle(){
		$email = $this->ask("Inserisci l'email dell'utente che vuoi
                                                    rendere revisore");
		$user = User::where('email, $email')->first();
```

```
        if(!$user){
                $this->error("Utente non trovato");
                return;
        }

        $user->is_revisor = true;
        $user->save();
        $this->info("L'utente {$user->name} e' ora un revisore.");
    }
```

$signature . Indichiamo quale comando Artisan vogliamo creare
$description . Descrizione del comando che comparira' quando facciamo *php artisan*
__construct . Costruttore che richiama la classe padre *Command*.
$email . Cattura la mail tramite ***$this->ask()*** di chi vogliamo far diventare *revisor*
$user . Cerchiamo nella tabella User se c'e' tale mail tramite il ***where()***.
if (!user) . Se l'utente non e' presente visualizza ***$this->error*** *(message)* e *return*
true . Assegnamo *true* ad is_revisor
info . Una volta fatto il ***save()*** visualizza questo messaggio.

In *app\Console\Kernel.php* che e' il file che governa sull'*Artisan:*

```
    protected $commands = [
            MakeUserRevisor::class
    ];
```

NB. Ricordiamoci di importare ***sempre*** le classi (*MakeUserRevisor, User ecc)*.

Dall'Artisan nella *Bash* ora visualizziamo il ***nuovo comando*** e completiamo i campi:

```
    php artisan presto:makeUserRevisor
```

- Creazione Middleware
Creiamo il Middleware in modo che solo gli utenti *Revisor* possano passare:

```
    php artisan make:middleware RevisorMiddleware
```

```
    class RevisorMiddleware{
            public function handle($request, Closure $next){

                    if(Auth::user() && Aauth::user()->is_revisor){
                            return $next($request);
                    }
```

```
                return redirect('/')->with('access.denied.revisor.only',
                                                        'access denied');
            }
        }
```

$request . Il middleware vara la richiesta
Closure . Altro blocco di codice che da il lascia passare al *RevisorController*
if() . Se l'utente e' *truthy* cioè esiste ed e' un utente *revisor* (ordine importante)
redirect . Se e' *Falsy* reindirizza con eventuale messaggio di errore.
'access.denied.revisor.only' . Nell'HTML *@if(session('access.denied.revisor.only'))*
inserisco messaggio di errore *class="alert alert-danger'*

Nel ***KernelHttp*** ho tutti i *Middleware* che scattano a prescindere, quelli che scattano dal file *Routing* web.php e tutte le rotte *api*.
Noi dobbiamo dare un nome al nostro Middleware.

In *app\Http\Kernel.php* creiamo il Middleware che protegge il nostro Controller:

```
protected $routeMiddleware = [
        ...
        'auth.revisor' => \App\Http|Middleware\RevisorMiddleware::class,
        ...
];
```

- Creazione Controller
Creiamo il RevisorController:

```
php artisan make:controller RevisorController
```

```
class RevisorController extends Controller{
        public function __construct(){
                $this->middleware('auth.revisor');
        }
}
```

middleware . Indichiamo il nostro middleware

E una ***route*** che ci indirizzi alla *index* del nostro controller: se l'utente e' revisore potrà visualizzarla. Aggiungiamo anche le rotte che ci consentono di ***accettare*** o ***eliminare*** un articolo.

```
Route::get('/revisor/home' [RevisorController::class, 'index'])->name('revisor.home');

Route::post('/revisor/announcement/{id}/accept', [RevisorController::class, 'accept']);
Route::post('/revisor/announcement/{id}/reject', [RevisorController::class, 'reject']);
```

- Creazione is_accepted
Con lo stesso procedimento di "***creazione is_revisor***" creiamo la colonna *is_accepted* nella tabella degli articoli.
Diamo ->***nullable*** al posto del default().

- Istruiamo il Controller

```
class RevisorController extends Controller{
	public function index(){
		$announcement = Announcement::where('is_accepted', null)
			->orderby('created_at', 'desc')
			->first();
		return view('revisor.home', compact('announcement');


	private function setAccepted($announcement_id, $value){
		$announcement = Announcement::find($announcement_id);
		$announcement->is_accepted = $value;
		$announcement->save();
		return redirect(route('revisor.home'));
	}

	public function accept($announcement_id){
		return $this->setAccepted($announcement_id, true);
	}

	public function reject($announcement_id){
		return $this->setAcceptec($announcement_id, false);
	}
}
```

where . Mostrami gli annunci non ancora revisionati, quindi *null*
setAccepted . Prendiamo l'*announcement_id* e dichiariamo se il valore e' *true* o *false*.
accept/reject() . Prendono l'*announcement_id* e lo fanno passare attraverso la funzione *setAccepted()*. Sono le funzioni collegate alle *Route*

- Tasto notifica Revisor

```
@ifguest
@else
@if(Auth::user()->is_revisor)
	<li class="nav-item">
		<a class="nav-link" href="{{route('revisor.home'}}">
			Revisor Home
		<span class="badge badge-pill badge-warning">
			{{App\Models\Announcement::ToBeRevisionedCount()}}</span>
		</a>
```

```
            </li>
        @endif
        @endguest
```

*Se l'utente e' un Revisor, mostrami il tasto Revisor Home con un badge che richiama alla classe statica **ToBeRevisionedCount()** che tiene il conto degli articoli da revisionare. Posizioniamolo nel **guest/endguest.***
Nel *Model* Announcement:

```
class Announcement extends Model{

        static public function ToBeRevisionedCount(){
              return Announcement::where('is_accepted', null)->count();
        }
}
```

. Creiamo il corpo dell'index
Creiamo il corpo dell'index con un'interfaccia che mi consenta di visualizzare per intero un *announcement* alla volta.

```
@if($announcement)
     <form action="{{route('revisor.reject', $announcement->id)}}"
                                                              method="POST">
      @csrf
      <button type="submit" class="btn btn-danger">Reject</button>
     </form>
@else
      <h3>Non ci sono piu' annunci da revisionare</h3>
@endif
```

form . Indirizzo il bottone Reject o Accept ad una rotta.
if/else . Se ci sono annunci, esegui il blocco, altrimenti *else* non ci sono piu' annunci.

- Istruiamo il PublicController

Nell'*index* e nella *categoryIndex* diciamo alla variabile che contiene i nostri annunci di visualizzare solo quelli che hanno un ***true*** nella colonna ***is_accepted.***

```
$announcements = Announcement::where('is_accepted', true)->orderBy.. //ecc
```