CHIAMATE AJAX

INSERIRE PIÙ IMMAGINI

Per assegnare più immagini ad un annuncio dobbiamo creare un nuovo *Model* e una nuova *Tabella* che contenga un *Record* per ogni immagine associata all'id dell'annuncio.
Il collegamento tra di loro sara' *One to Many*.

Dal momento che l'annuncio non e' ancora stato creato, il server non sa a che *id annuncio* associarsi.
Per far che ciò avvenga facciamo una chiamata ***Ajax*** che permette di effettuare questa associazione in un tempo successivo.

Nella comunicazione tra *Client* e *Server* si crea automaticamente un *codice segreto (**reference**)* che verrà associato tramite la chiamata *Ajax* (che e' una chiamata che non fa ricaricare la pagina, ma che si muove su lato *Javascript*) e *Dropzone* (un elemento Javascript opportunamente inizializzato) alle immagini che inseriamo nel *form* e che fino al *submit* della richiesta verranno tenute in attesa.

Al momento della richiesta *POST* l'*id* dell'annuncio e la *reference* metteranno in associazione *immagini* e *annuncio*.

Per creare un *codice segreto **casuale*** possiamo utilizzare una funzione.
Questa ci permetterà di avere un ID unico in hash a base 36 caratteri.

- Nel *controller,* nella funzione che mostra il form di creazione dell'annuncio:

```
public function newAnnouncement(Request $request){
      $uniqueSecret = $request->old('uniqueSecret',
            base_convert(sha1(uniqid(mt_rand())), 16,36));
      return view('announcements.new', compact('uniqueSecret'));
}
```

old . Nel caso di errore di validazione della Request quando carichiamo l'annuncio, restituisci il vecchio *uniqueSecret***.**

Nella relativa *view*:

```
<form action="{{route('announcement.create')}}" method="POST">
@csrf
      <input type="hidden" name="uniqueSecret" value"{{$uniqueSecret}}">
...
```

Dobbiamo adesso utilizzare le ***librerie Dropzone*** per catturare questa chiamata *Ajax* e inserire le immagini nella tabella apposita.

Link - https://www.dropzonejs.com/

- Nella *bash*:

```
npm i dropzone --save-dev
```

- In *app.scss* importiamo la libreria appena installata:

```
@import '~dropzone/dist/basic':
@import '~dropzone/dist/dropzone';
```

- In *app.js*:

```
document.Dropzone = require('dropzone');
Dropzone.autoDiscover = false;
require('./announementImages');
```

*Abbiamo costruito una variabile dell'oggetto generale del nostro browser **window** chiamata**.Dropzone** che conterrà la libreria (**funzione costruttore**).*
*Con **autoDiscover** diciamo a questa libreria che vogliamo gestire noi l'aggancio ad un elemento Html e aggiungiamo **false**.*

- Buildiamo gli *assets*:

```
npm run prod
```

- Inseriamo nel *form* il box che conterrà il *javascript* che ci permette di caricare le immagini:

```
<div class="form-group row">
      <label for="images" class="col--md-12 col-form-label
                                          text-md-right">Immagini</label>
      <div class="col-md-12">

            <div class="dropzone" id="drophere"></div>

            @error('images')
                  <span class="invalid-feedback" role="alert">
                        <strong>{{$message}}</strong>
                  </span>
            @enderror
      </div>
</div>
```

- Creiamo del *Javascript* da inserire al suo interno tramite il *building degli assets*.
NB: Per farlo creiamo un nuovo file Js collegato all'*app.js*

```
$(function(){
	if ($('#drophere").length > 0 ){
		let csrfToken = $('meta[name="csrf-token"]').attr('content');
		let uniqueSecret =
				$('input[name="uniqueSecret"]').attr('value');

		let myDropzone = new Dropzone('#drophere', {
			url:'/announcement/images/upload',

			params:{
				_token: csrfToken,
				uniqueSecret: uniqueSecret
			}
		})
	// alert ('ci sono');
	}
})
```

$. Chiamata JQuery.
alert . Messaggio che utilizziamo a scopo di programmazione per controllare che il Js risponda ai comandi
if($('#drophere") . Diciamo alla funzione di scattare solo quando c'e' l'Id selezionato nella pagina. Se e' quindi *true*, ossia maggiore di 0.
new Dropzone . Inizializziamo un nuovo elemento Dropzone agganciato a ***#drophere***
url . Il percorso che deve fare (tramite richiesta *POST*) per inviare l'immagine che e' stata appena trascinata nel box.
let csrfToken . Catturiamo il *csrf Token* in una variabile attraverso un elemento ***meta*** per permettere successivamente alla immagine di concludere con successo la richiesta POST che effettuiamo per trasportare l'immagine (***_token***).
uniqueSecret . Invia anche il *codice segreto* che abbiamo isolato anch'esso in una variabile.

- Creiamo una *Rotta* relativa all'***url*** specificato nel *Js:*

```
Route::post('/announcement/images/upload' [HomeController::class,
'uploadImage'])->name('announcement.images.upload)
```

-Nel Controller sviluppiamo la funzione:

```
public function uploadImage(Request $request){
        //dd($request->input()); 1.
        $uniqueSecret = $request->input('uniqueSecret');
        $fileName = $request->file('file')->store("public/temp/{$uniqueSecret}");
        session()->push("images.{$uniqueSecret}", $fileName);

        return response()->json(
//session()->get("images.{$uniqueSecret}));
        [
        'id' =>$fileName
        ]
}
```

NB. Importiamo la classe *use Illuminate\Http\Request;*
dd()1 . Ispezionando il sito, quando carichiamo l'immagine, nella sezione *network* possiamo controllare che l'invio avvenga correttamente.
$uniqueSecret . Catturiamo il *codice segreto* in una variabile.
->store() . Utilizziamo la funzionalità' *Storage* di Laravel per memorizzare su disco ***temp/{$codSeg}*** il file contenuto nella request.
session() . Aggiungiamo in una *variabile di sessione* tramite ***push()*** (che mette in coda le richieste), il ***$fileName*** appena creato, e lo nominiamo ***images.{$codseg}***
response()->json() . Diamo una risposta di tipo *json*
session()->get(...) . Catturiamo nel *json* la immagine che prendiamo ***(get)*** dalla *session.*
'id' . Catturo l'*id* nella ***response*** su *Javascript*.

Nel Network possiamo vedere l'avanzamento del passaggio dati, ogni volta che carichiamo un'immagine.

Le immagini in questo momento sono nella cartella *storage/app/public/temp/cartellaCodiceSegreto.*

NB: Lanciamo *npm storage:link*

- Creiamo il *Model* AnnouncementImage nella *bash*:

```
php artisan make:model AnnouncementImage -m
```

- Istruiamo la Migration con all'interno la *foreign key*:

```
public function up(){
        Schema::create('announcement_images', function (Blueprint $table){
                $table->bigIncrements('id');
                $table->string('file');
                $table->unsignedBigInteger('announcement_id');

$table->foreign('announcement_id')->references('id')->on('announcements');
                $table->timestamps():
        });
```

```
    }
```

NB. Nella *Bash* eseguiamo la *Migration*.

- Istruiamo il *Model* appena creato e relazioniamolo all'altro *Model*:

```
        class AnnouncementImage extends Model{
               public function announcement(){
                    return $this->belongsTo(Announcement::class);
              }
        }


class Announcement extends Model{
       public function images(){
             return $this->hasMany(AnnouncementImage::class);
}
```

- Nel Controller dopo il *Mass Assignment* per la creazione dell'annuncio, *catturo* il codice segreto e creiamo i record del nuovo *Model* appena creato.

```
      public function createAnnouncement(AnnouncementRequest $request){

      ... //creiamo l'annuncio come di consueto

           $uniqueSecret = $request->input('uniqueSecret');

           $images = session()->get("images.{$uniqueSecret}", [ ] );
           $removedImages = session()->get("removedimages.{$uniqueSecret}", [ ] );

           $images = array_diff($images, $removedImages);

           foreach($images as $image){
               $i = new AnnouncementImage();

               $fileName = basename($image);
               $newFileName = "public/announcements/{$a->id}/{$fileName}";
               Storage::move($image, $newFileName);

               $i->file = $newFileName;
               $i->announcement_id = $a->id;

               $i->save();
           }

      File::deleteDirectory(storage_path("/app/public/temp/{$uniqueSecret}"));

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

$images . Prendiamo le ***images*** caricate nella sessione (come scritto nel Js) e facciamo un ***foreach***

$removedImages . Prendiamo le ***images*** nella sessione delle ***removedimages*** e nel caso non ce ne fossero deve restituire un ***[ ]*** *array vuoto.*

array_diff . Mettiamo in ***$images*** solo le images di sessione che non sono state rimosse.

$i . Creiamo per ogni immagine una variabile contenente un nuovo Record.

basename . Tramite questa funzione di Laravel posso andare a catturare solo il nome della singola *$image* assegnandolo alla variabile ***$fileName***

Storage::move . Utilizzo la *facade Storage* per spostare il file dalla cartella *temp* alla cartella *public/announcements/* seguita da ***{$a->id)*** che e' l'*ID* dell'annuncio creato ad inizio della funzione *createAnnouncement,* in modo di averle tutte raggruppate in una singola cartella, e a seguire il loro ***$fileName***.

$i->file . Assegno al *record* dell' *$image* appena creato il ***$file*** (corrispondenza con Migrate)

$i->announcement_id . Assegno al *record* dell'*$image* appena creato l'*id* dell'Annuncio (**Foreign Key**)

File::delete.. . Una volta completato il *foreach* elimina la cartella *Temp* che conteneva i File prima del loro salvataggio.

- Aggiungiamo ad *AnnouncementImages.js* della Logica per inserire un tasto ***remove*** alle immagini che carichiamo nel caso cambiassimo idea:

```
...
params:{
                    _token: csrfToken,
                    uniqueSecret: uniqueSecret
            },

            addRemoveLinks:true

      });

      myDropzone.on("success", function(file, response){
            file.serverId = response.id;
      };

      myDropzone.on("removedfile", function(file){
            $ajax({
                  type: 'DELETE',
                  url: '/announcement/images/remove'
                  data: {
                        _token: csrfToken,
                        id: file.serverId,
                        uniqueSecret: uniqueSecret
                  },
                  dataType: 'json'
            });
      });
})
```

addRemoveLinks . Diciamo a Dropzone che vogliamo il bottoncino *remove* sotto l'immagine

response.id . Ci agganciamo all'evento ***success*** di Dropzone che scatta quando l'immagine viene caricata sul server. Catturiamo quindi in un *field* ***file.serverId*** la risposta del server.

ajax . Facciamo una chiamata *Ajax* di tipo Delete per eliminare in caso di *remove* e passiamo il codice ***file.serverId*** che corrisponde all'immagine appena scelta dall'utente.

- Creiamo la *Route delete* corrispondente alla chiamata *Ajax:*

```
Route::delete('/announcement/images/remove', [HomeController::class ,
'removeImage'])->name('announcement.images.remove')
```

- Creiamo la funzione nel Controller:

```
public function removeImage(Request $request){
        $uniqueSecret = $request->input('uniqueSecret');
        $fileName = $request->input('id');
        session()->push("removedimages.{$uniqueSecret}", $fileName);
        Storage::delete($fileName);

        return response()->json('ok');
}
```

$fileName **.** Catturiamo il FileName che e' l'Id specificato nella funzione *UploadImage.*

$session()->push(...) . Creiamo un nuovo *array in sessione* chiamato ***removedimages*** e aggiungiamo tutte le immagini (***$fileName***) che l'utente vuole rimuovere.

Storage::delete . Successivamente eliminiamo dallo storage *temp* le Immagini.

- Creiamo una nuova *Route* che gestirà la restituzione delle immagini già inserite in caso di non superamento della validazione del Form.

```
Route::get('/announcement/images', [HomeController::class,
'getImages'])->name('announcement.images');
```

- Nella rispettiva *funzione* nel *controller*:

```
    public function getImages(Request $request){
        $uniqueSecret = $request->input('uniqueSecret');
        $images = session()->get("images.{$uniqueSecret}", [ ] );
        $removedImages = session()->get("removedimages.{$uniqueSecret}", [ ]
    );
        $images = array_diff($images, $removedImages);
        $data = [ ] ;
        foreach($images as $image) {
            $data[ ] = [
                'id' => $image,
                'src' => Storage::url($image)
            ];
        }
        return response()->json($data);
    }
```

$data . Conterrà l'*id* e la *url* per visualizzare il file.
json . Spediamo il *$data* al *json* per ricevere poi da Dropzone le foto in risposta.

Se refresho l'immagine avendo gia' caricato delle immagini, non le vedro', ma saranno caricate comunque. Per vederle bel browser inserisco l'URI della rotta appena creata e aggiungo ?uniqueSecret=xxxxxxx

- Per permettere a *Dropzone* di mostrarmi le immagini che gia' sono state caricate, istruisco il file *javascript aggiungendo*

```
        adRemoveLinks: true,

        init: function() {
            $.ajax({
                type: 'GET',
                url: '/announcement/images',
                data: {
                    uniqueSecret: uniqueSecret
                },
                dataType: 'json'
            }).done(function(data){
                $.each(data, function(key, value){
                    let file = {
                        serverId: value.id
                    };

                    myDropzone.options.addedfile.call(myDropzone, file);
                    myDropzone.options.thumbnail.call(myDropzon, file, value.src);
                });
            });
        }
});
```

Init . All'inizializzazione lancia questa funzione. Una chiamata *Ajax* di tipo *GET*
done. Quando questa operazione sarà conclusa metti quelle informazioni in ***data***.

Grazie a jquery, tradurrà la risposta del json in un file javascript che posso ciclare tramite ***each****. Ogni data conterrà in key id e src dell'immagine.*
Successivamente diciamo a MyDropzone (il box nel form) di aggiungere ***call*** *il file contenente il valore dell'id, e il src che contiene l'immagine stessa.*

- Inseriamo il codice della *view* per visualizzare le immagini:

```
@foreach($announcement->images as $image)
	<div class="row mb-2">
		<div class="col-md-4">
			<img src={{Storage::url($image->file)}}"
class="rounded" alt="">
		</div>
		<div class ="col-md-8>
			{{$image->id}} <br>
			{{$image->file}} <br>
			{{Storage::url($image->file)}} <br>
		</div>
	</div>
@endforeach
```