The Aulab Post L11 - USER STORY #5

User Story #5:

- **Come** Corrado
- **vorrei** poter gestire in autonomia tags e categorie,
- **in modo tale** da avere una piattaforma sempre aggiornata

ACCEPTANCE CRITERIA:

- Creazione dei Tags
- La relazione tra Tags e Articoli è N-a-N
- Permettere solo all'admin modifica e cancellazione dei tags
- Permettere solo all'admin la modifica e cancellazione delle categorie

EXTRA:

- Richiamare i tags nei metadati della pagina di dettaglio

USER STORY 5 - TAGS E CATEGORIE DINAMICHE

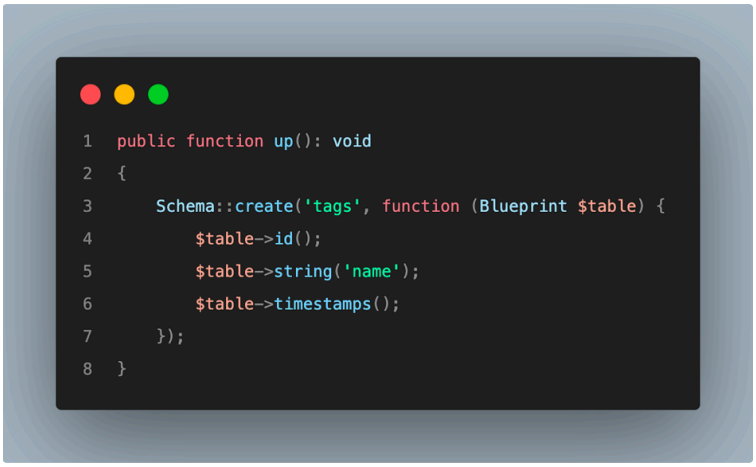
CREAZIONE MODELLO TAG E RELAZIONI

In questa User Story, daremo la possibilità all'**utente Writer di inserire dei tags** per ogni articolo. Daremo poi invece all'**utente Admin** completa libertà su questi tag, ovvero di **cancellarli** o **modificarli**.

Prima di tutto, creiamo il **modello** dedicato, insieme alla sua **migrazione** per gestire i **Tag**. Da terminale lanciamo:

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

Andiamo a gestire la migrazione `create_tags_table`:



Per confermare le modifiche sul DB, lanciamo la classica serie di comandi da terminale:

```
php artisan migrate
php artisan migrate:rollback
```

```
php artisan migrate
```

Aggiungiamo ora il **fillable** nel modello **Tag**:

```php
class Tag extends Model
{
    ...
    protected $fillable = ['name'];
}
```

I **tags saranno relazionati agli articoli** tramite una relazione **Many-to-Many**: un articolo infatti potrà avere più tags e un tag potrà essere utilizzato in più articoli *(Tag N : N Article).* Il primo passo per gestire una relazione di questo tipo è creare una **tabella pivot**, ovvero una tabella che si frappone fra le due tabelle da relazionare: in questo modo sarà molto semplice per Laravel (e per noi) seguire tutti gli elementi relazionati. Per convenzione di Laravel, questa tabella avrà come nome il **nome dei due modelli relazionati, al singolare e in ordine alfabetico**.

Da terminale, lanciamo il comando per creare la migrazione per tabella:

```
php artisan make:migration create_article_tag_table
```

Questa tabella conterrà **due Foreign Keys**, una che farà riferimento all'**articolo** e l'altra invece al **tag**:

```php
public function up(): void
{
    Schema::create('article_tag', function (Blueprint $table) {
        $table->id();
        $table->unsignedBigInteger('article_id');
        $table->foreign('article_id')->references('id')->on('articles');
        $table->unsignedBigInteger('tag_id');
        $table->foreign('tag_id')->references('id')->on('tags');
        $table->timestamps();
    });
}
```

Una volta terminato, lanciamo la serie di comandi:

```
php artisan migrate
php artisan migrate:rollback
php artisan migrate
```

Una volta gestito il database, dobbiamo anche gestire Eloquent, andando a creare all'**interno dei modelli le funzioni di relazione**. Quindi, in `App\Models\Article.php` scriveremo:

```php
...
public function tags(){
    return $this->belongsToMany(Tag::class);
}
```

E in `App\Models\Tag.php` scriveremo:

```php
public function articles(){
    return $this->belongsToMany(Article::class);
}
```

INSERIMENTO DEI TAGS NEGLI ARTICOLI

Il prossimo passo è quello di permettere ai **redattori di inserire i tags** relativi all'articolo che stanno scrivendo.Andiamo quindi ad aggiornare il form in `resources/views/article/create.blade.php`:

```blade
...
<div class="mb-3">
    <label for="tags" class="form-label">Tags</label>
    <input type="text" name="tags" class="form-control" id="tags" value="{{old('tags')}}">
    <span class="small text-muted fst-italic">Dividi ogni tag con una virgola</span>
    @error('tags')
        <span class="text-danger">{{$message}}</span>
    @enderror
</div>
...
```

Nell'**ArticleController**, nella funzione `store()` andremo quindi ad aggiornare in questo modo:

```php
public function store(Request $request)
{
    $request->validate([
        'title' => 'required|unique:articles|min:5',
        'subtitle' => 'required|min:5',
        'body' => 'required|min:10',
        'image' => 'required|image',
        'category' => 'required',
        'tags' => 'required'
    ]);

    $article = Article::create([
        'title' => $request->title,
        'subtitle' => $request->subtitle,
        'body' => $request->body,
        'image' => $request->file('image')->store('public/images'),
        'category_id' => $request->category,
        'user_id' => Auth::user()->id,
    ]);

    $tags = explode(',', $request->tags);

    foreach($tags as $i => $tag){
        $tags[$i] = trim($tag);
    }

    foreach($tags as $tag){
        $newTag = Tag::updateOrCreate([
            'name' => strtolower($tag)
        ]);
        $article->tags()->attach($newTag);
    }
    return redirect(route('homepage'))->with('message', 'Articolo creato con successo');
}
```

Importare:

```php
use App\Models\Tag;
```

Cosa abbiamo fatto?

1. Abbiamo inserito i **tags all'interno delle regole di validazione**;
2. Tramite la funzione `explode()` abbiamo ottenuto **da una stringa** (l'input text "tags") **un array di elementi**. Questa funzione accetta due parametri: il primo è il divisore (nel nostro caso, `,` quindi la virgola) e il secondo è la stringa da dividere;
3. Una volta ottenuto l'array di tags, abbiamo lanciato la funzione `updateOrCreate()` la quale automaticamente fa un controllo nel database: **se il tag con quel nome non esiste, lo creerà, e con** `strtolower` cambia i caratteri tutti in minuscolo; in caso contrario, farà un semplice update. In questo modo, la nostra tabella tags sarà pulita e senza ripetizioni;
4. Una volta gestito ogni singolo tag, andremo a lanciare la funzione `attach()`, che è la funzione che crea effettivamente il record all'interno della tabella pivot e che, quindi, **crea la relazione**.

Il prossimo passo è quello di mostrarli nelle card in **homepage**, **index**, **search-index, by-category, by-user**, e nella pagina **show,** ciclando i tag relazionati all'articolo.

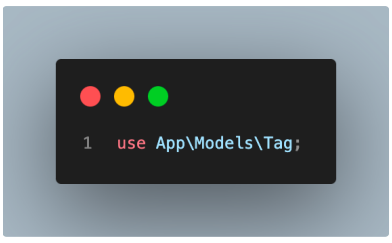
```
<p class="small text-muted my-0">
    @foreach ($article->tags as $tag)
        #{{ $tag->name }}
    @endforeach
</p>
```

GESTIONE DEI TAG LATO ADMIN

Adesso che abbiamo permesso al redattore di inserire dei tags alla creazione dell'articolo, diamo la possibilità a tutti gli utenti **admin di tenere sotto traccia la situazione**. Prima di tutto, facciamo in modo che tutti i record salvati nella tabella tags siano a disposizione automaticamente di tutte le viste del nostro progetto. Andiamo quindi in `App\Providers\AppServiceProvider.php` e scriviamo:

```
public function boot(): void
{
    ...
    if(Schema::hasTable('tags')){
        $tags = Tag::all();
        View::share(['tags' => $tags]);
    }
}
```

Importiamo il modello Tag:



```
use App\Models\Tag;
```

Andremo, quindi ad **aggiornare la dashboard dell'admin con le nuove informazioni**. Gestiamo anche questo elemento come se fosse un componente. In `resources/views/components` creiamo un file chiamato `metainfo-table.blade.php`.

Adesso richiamiamo questo componente nella **dashboard dell'admin**. **Useremo questo componente per gestire sia i tags che le categorie**: quindi a questo componente passeremo non solo tutti i record delle rispettive tabelle, ma anche una stringa dove andremo a specificare se stiamo modificando tags o categorie. In questo modo potremo sfruttare il dato per un controllo.

Partiamo con i tag:

```blade
<hr>
<div class="container my-5">
    <div class="row justify-content-center">
        <div class="col-12">
            <h2>Tutti i tags</h2>
            <x-metainfo-table :metaInfos="$tags" metaType="tags"/>
        </div>
    </div>
</div>
```

Nel **componente**, invece, inseriamo una **tabella.** Al suo interno creiamo dei form, uno per la modifica e uno per la cancellazione del tag.

```blade
<table class="table table-striped table-hover">
    <thead class="table-dark">
        <tr>
            <th scope="col">#</th>
            <th scope="col">Nome tag</th>
            <th scope="col">Q.tà articoli collegati</th>
            <th scope="col">Aggiorna</th>
            <th scope="col">Cancella</th>
        </tr>
    </thead>
    <tbody>
        @foreach ($metaInfos as $metaInfo)
            <tr>
                <th scope="row">{{$metaInfo->id}}</th>
                <td>{{$metaInfo->name}}</td>
                <td>{{count($metaInfo->articles)}}</td>
                @if ($metaType == 'tags')
                    <td>
                        <form action="">
                            @csrf
                            @method('PUT')
                            <input type="text" name="name" placeholder="Nuovo nome tag" class="form-control w-50 d-inline">
                            <button type="submit" class="btn btn-secondary">Aggiorna</button>
                        </form>
                    </td>
                    <td>
                        <form action="">
                            @csrf
                            @method('DELETE')
                            <button type="submit" class="btn btn-danger">Elimina</button>
                        </form>
                    </td>
                @endif
            </tr>
        @endforeach
    </tbody>
</table>
```

Dobbiamo quindi adesso creare la logica per entrambe le possibilità. Partiamo dalla modifica. Come sempre, **partiamo dalla rotta in web.php**

```php
Route::middleware('admin')->group(function(){
    ...
    Route::put('/admin/edit/tag/{tag}', [AdminController::class, 'editTag'])->name('admin.editTag');
});
```

Creiamo la funzione nell'**AdminController**:

```php
public function editTag(Request $request, Tag $tag){
    $request->validate([
        'name' => 'required|unique:tags',
    ]);
    $tag->update([
        'name' => strtolower($request->name),
    ]);
    return redirect()->back()->with('message', 'Tag aggiornato correttamente');
}
```

Importiamo il Tag:

```php
use App\Models\Tag;
```

Tra le regole abbiamo specificato non solo che è necessario inserire un valore, ma anche che nella tabella non ci può essere un altro tag con lo stesso nome. Poi, prendiamo ciò che inserisce l'utente nell'input, lo trasformiamo in lowercase e andiamo ad aggiornare il tag già presente. Fatto questo, possiamo aggiornare il form, includendo anche una logica di override del metodo. Andiamo adesso a modificare il primo form del componente **metainfo-table**:

```blade
@if ($metaType == 'tags')
    <td>
        <form action="{{route('admin.editTag', ['tag' => $metaInfo])}}" method="POST">
            @csrf
            @method('PUT')
            <input type="text" value="{{$metaInfo->name}}" name="name" placeholder="Nuovo nome tag" class="form-control w-50 d-inline">
            <button type="submit" class="btn btn-secondary">Modifica</button>
        </form>
    </td>
    ...
```

Facciamo qualche test e poi **occupiamoci di gestire la cancellazione**. Partiamo, come sempre, dalla **rotta in web.php**:

```php
Route::middleware('admin')->group(function(){
    ...
    Route::delete('/admin/delete/tag/{tag}', [AdminController::class, 'deleteTag'])->name('admin.deleteTag');
});
```

Spostiamoci nell'**AdminController** e gestiamo il metodo:

```php
public function deleteTag(Tag $tag){
    foreach($tag->articles as $article){
        $article->tags()->detach($tag);
    }
    $tag->delete();
    return redirect()->back()->with('message', 'Tag eliminato correttamente');
}
```

Prima di cancellare il tag, dobbiamo gestire il vincolo d'integrità referenziale. Questo vuol dire che prima dobbiamo sciogliere ogni relazione tra il tag scelto ed eventuali articoli, e solo dopo possiamo cancellare il tag. Ora che abbiamo gestito la logica, **spostiamoci nel blade del componente** per modificare il secondo form:

```blade
    ...
    <td>
        <form action="{{route('admin.deleteTag', ['tag' => $metaInfo])}}" method="POST">
            @csrf
            @method('DELETE')
            <button type="submit" class="btn btn-danger">Elimina</button>
        </form>
    </td>
@endif
```

GESTIONE DELLE CATEGORIE LATO ADMIN

Abbiamo creato prima un componente che permette all'amministratore di gestire le meta-informazioni. Richiamiamo dunque nuovamente il componente in `resources/views/admin/dashboard.blade.php`, ma **questa volta passiamogli le categorie**:

```
...
<div class="container my-5">
    <div class="row justify-content-center">
        <div class="col-12">
            <h2>Tutte le categorie</h2>
            <x-metainfo-table :metaInfos="$categories" metaType="categorie"/>
        </div>
    </div>
</div>
```

Aggiungiamo anche altri due form nel componente **metainfo-table** nella direttiva `@else`**:**

```
...
@else
    <td>
        <form action="">
            @csrf
            @method('PUT')
            <input type="text" name="name" placeholder="Nuovo nome categoria" class="form-control w-50 d-inline">
            <button type="submit" class="btn btn-secondary">Aggiorna</button>
        </form>
    </td>
    <td>
        <form action="">
            @csrf
            @method('DELETE')
            <button type="submit" class="btn btn-danger">Elimina</button>
        </form>
    </td>
@endif
```

Adesso andiamo a gestire la logica di modifica categoria, partendo come sempre dalla **rotta in web.php**:

```
Route::middleware('admin')->group(function(){
    ...
    Route::put('/admin/edit/category/{category}', [AdminController::class, 'editCategory'])->name('admin.editCategory');
});
```

Andiamo ora nell'**AdminController** a creare la funzione:

```php
public function editCategory(Request $request, Category $category){
    $request->validate([
        'name' => 'required|unique:categories',
    ]);
    $category->update([
        'name' => strtolower($request->name),
    ]);
    return redirect()->back()->with('message', 'Categoria aggiornata correttamente');
}
```

Importare Category:

```php
use App\Models\Category;
```

La logica è la medesima utilizzata per l'aggiornamento dei tags. Passiamo ad aggiornare quindi il componente `resources/views/components/metainfo-table.blade.php`:

```
@else
    <td>
        <form action="{{route('admin.editCategory', ['category' => $metaInfo])}}" method="POST">
            @csrf
            @method('PUT')
            <input type="text" value="{{$metaInfo->name}}" name="name" placeholder="Nuovo nome categoria" class="form-control w-50 d-inline">
            <button type="submit" class="btn btn-secondary">Aggiorna</button>
        </form>
    </td>
    ...
```

Occupiamoci adesso della cancellazione, partendo sempre dalla **rotta in web.php**:

```php
Route::middleware('admin')->group(function(){
    ...
    Route::delete('/admin/delete/category/{category}', [AdminController::class, 'deleteCategory'])->name('admin.deleteCategory');
});
```

Spostiamoci nell'**AdminController**:

```php
public function deleteCategory(Category $category){
    $category->delete();
    return redirect()->back()->with('message', 'Categoria eliminata correttamente');
}
```

Questa volta non c'è stato bisogno di gestire il vincolo d'integrità referenziale perché ne abbiamo gestito il comportamento direttamente nelle migrazioni. Passiamo ad aggiornare quindi il componente `resources/views/components/metainfo-table.blade.php`:

```php
    ...
    <td>
        <form action="{{route('admin.deleteCategory', ['category' => $metaInfo])}}" method="POST">
            @csrf
            @method('DELETE')
            <button type="submit" class="btn btn-danger">Elimina</button>
        </form>
    </td>
@endif
```

Abbiamo quindi completato la gestione delle categorie. La possibilità di cancellare una categoria però può mostrare il fianco ad alcuni problemi: nelle card dei nostri articoli, infatti, noi abbiamo un link che utilizza proprio la categoria relazionata a un articolo.

Per ovviare al problema, abbiamo bisogno di un semplice `if` all'interno delle nostre card presenti in `welcome.blade.php`, `index.blade.php`, `search-index.blade.php`, `by-category.blade.php`, `by-user.blade.php`:

```php
@if ($article->category)
    <p class="small text-muted">Categoria:
        <a href="{{route('article.byCategory', $article->category)}}" class="text-capitalize text-muted">{{ $article->category->name }}</a>
    </p>
@else
    <p class="small text-muted">Nessuna categoria</p>
@endif
<p class="small text-muted my-0">
    @foreach ($article->tags as $tag)
        #{{ $tag->name }}
    @endforeach
</p>
```

Lo stesso anche per la vista show.blade.php

```php
@if ($article->category)
    <p class="fs-5">Categoria:
        <a href="{{route('article.byCategory', $article->category)}}" class="text-capitalize fw-bold text-muted">{{ $article->category->name }}</a>
    </p>
@else
    <p class="fs-5">Nessuna categoria</p>
@endif
```

CREAZIONE DI NUOVE CATEGORIE

Diamo all'utente amministratore anche la possibilità di **creare delle nuove categorie**. Partiamo facendo la rotta di creazione in **web.php** nel gruppo con il **middleware admin**:

```php
Route::middleware('admin')->group(function(){
    ...
    Route::post('/admin/category/store', [AdminController::class, 'storeCategory'])->name('admin.storeCategory');
});
```

Nell'**AdminController**, creiamo la funzione:

```php
public function storeCategory(Request $request){
    Category::create([
        'name' => strtolower($request->name),
    ]);
    return redirect()->back()->with('message', 'Categoria inserita correttamente');
}
```

Aggiungiamo adesso un nuovo form nella vista `resources/views/admin/dashboard.blade.php`, **nella sezione dedicata alle categorie**:

```blade
...
<div class="d-flex justify-content-between">
    <h2>Tutte le categorie</h2>
    <form action="{{route('admin.storeCategory')}}" method="POST" class="w-50 d-flex m-3">
        @csrf
        <input type="text" name="name" class="form-control me-2" placeholder="Inserisci una nuova categoria">
        <button type="submit" class="btn btn-outline-secondary">Inserisci</button>
    </form>
</div>
<x-metainfo-table :metaInfos="$categories" metaType="categorie"/>
...
```

Abbiamo completato la **quinta User Story**, non ci resta che testare tutto e successivamente pushare:

```
git add .
git commit -m "User Story 5 completata"
git push
```

Fine User Story 5
