

The Aulab Post L11 - USER STORY #6

User Story #6: [🔗](#)

- **Come** Sara
- **vorrei** poter cancellare o modificare gli articoli che ho scritto
- **in modo tale** da consegnare sempre un lavoro d'alta qualità

ACCEPTANCE CRITERIA: [🔗](#)

- Creazione di una dashboard dedicata ai writer
- Permettere solo al writer specifico la modifica di un articolo
- Permettere solo al writer specifico la cancellazione di un articolo

EXTRA: [🔗](#)

- Se l'articolo viene modificato, deve ritornare alla revisione
- Se modifico l'immagine di un articolo, cancellare la vecchia immagine dallo Storage
- Quando l'articolo viene cancellato, cancellare anche l'immagine dallo Storage

USER STORY 6 - WRITER DASHBOARD [🔗](#)

CREAZIONE DELLA DASHBOARD [🔗](#)

In questa User Story, diamo l'opportunità ai **redattori di modificare e cancellare gli annunci che hanno scritto** personalmente. Andiamo quindi, prima di tutto, a **creare una dashboard** dove l'**utente redattore** può visualizzare tutti i suoi articoli.

Creiamo un nuovo controller con il seguente comando da terminale:

```
1 php artisan make:controller WriterController
```

Per creare la nostra logica, partiamo come sempre dalla **rotta in web.php** da inserire nel gruppo di rotte con **middleware writer** :



```
1 Route::middleware('writer')->group(function(){
2     ...
3     Route::get('/writer/dashboard', [WriterController::class, 'dashboard'])->name('writer.dashboard');
4 });
```

Importiamo il controller:

```
1 use App\Http\Controllers\WriterController;
```

Andiamo nel **WriterController** e creiamo la funzione:

```
1 public function dashboard(){
2     $articles = Auth::user()->articles()->orderBy('created_at', 'desc')->get();
3     $acceptedArticles = $articles->where('is_accepted', true);
4     $rejectedArticles = $articles->where('is_accepted', false);
5     $unrevisionedArticles = $articles->where('is_accepted', NULL);
6
7     return view('writer.dashboard', compact('acceptedArticles', 'rejectedArticles', 'unrevisionedArticles'));
8 }
```

Importiamo:

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

Quello che stiamo facendo, in questo caso, è dividere in tre variabili diversi gli articoli **accettati**, **rifiutati** e **in fase di revisione** dell'utente writer attualmente loggato. E' una logica simile a quella applicata alla dashboard del revisore. Per gli articoli accettati e rifiutati abbiamo utilizzato il metodo where()

All'interno di `resources/views/` creiamo una nuova directory chiamata `writer` e al suo interno creiamo un file chiamato `dashboard.blade.php`. Anche in questo file avremo tre tabelle, con al loro interno gli articoli accettati, rifiutati o da revisionare.

Creiamo nella cartella `resources/views/components` un file chiamato `writer-articles-table.blade.php`, dove inseriamo una **tabella**. Nella vista della **dashboard del redattore**, richiamiamo tre volte il componente della **tabella writer-articles**, passandogli ogni volta l'articolo a seconda se è stato accettato o meno:

```

1 <x-layout>
2   <div class="container-fluid p-5 bg-secondary-subtle text-center">
3     <div class="row justify-content-center">
4       <div class="col-12">
5         <h1 class="display-1">Bentornato, Redattore {{Auth:user()->name}}</h1>
6       </div>
8     </div>
9     <div class="container my-5">
10      <div class="row justify-content-center">
11        <div class="col-12">
12          <h2>Articoli in attesa di revisione</h2>
13          <x-writer-articles-table :articles="$unrevisedArticles"/>
14        </div>
15      </div>
16    </div>
17    <div class="container my-5">
18      <div class="row justify-content-center">
19        <div class="col-12">
20          <h2>Articoli pubblicati</h2>
21          <x-writer-articles-table :articles="$acceptedArticles"/>
22        </div>
23      </div>
24    </div>
25    <div class="container my-5">
26      <div class="row justify-content-center">
27        <div class="col-12">
28          <h2>Articoli respinti</h2>
29          <x-writer-articles-table :articles="$rejectedArticles"/>
30        </div>
31      </div>
32    </div>
33 </x-layout>

```

All'interno di questo componente, faremo visualizzare al redattore tutti i dettagli degli articoli che ha scritto. Quindi, modifichiamo il file della tabella: per ogni stato dell'articolo **cicliamo** le righe della tabella mostrando i rispettivi dati. Inoltre, inseriamo anche tre bottoni per poter visualizzare, modificare ed eliminare l'articolo.

```

1 <table class="table table-striped table-hover">
2   <thead class="table-dark">
3     <tr>
4       <th scope="col">#</th>
5       <th scope="col">Titolo</th>
6       <th scope="col">Sottotitolo</th>
7       <th scope="col">Categoria</th>
8       <th scope="col">Tags</th>
9       <th scope="col">Inserito il</th>
10      <th scope="col">Azioni</th>
11    </tr>
12  </thead>
13  <tbody>
14    @foreach ($articles as $article)
15      <tr>
16        <th scope="row">{{ $article->id }}</th>
17        <td>{{ $article->title }}</td>
18        <td>{{ $article->subtitle }}</td>
19        <td>{{ $article->category->name ?? 'Nessuna categoria' }}</td>
20        <td>
21          @foreach ($article->tags as $tag)
22            #{{ $tag->name }}
23          @endforeach
24        </td>
25        <td>{{ $article->created_at->format('d/m/Y') }}</td>
26        <td>
27          <a href="{{ route('article.show', $article) }}" class="btn btn-secondary">Leggi</a>
28          <a href="#" class="btn btn-warning text-white">Modifica</a>
29          <form action="#" method="#" class="d-inline">
30            <button type="submit" class="btn btn-danger">Elimina</button>
31          </form>
32        </td>
33      </tr>
34    @endforeach
35  </tbody>
36 </table>

```

Adesso dobbiamo dare però la possibilità all'utente redattore di raggiungere questa dashboard. Andiamo quindi nel dropdown della `navbar.blade.php` :

```

1 ...
2 @if (Auth::user()->is_writer)
3   <li><a class="dropdown-item" href="{{ route('writer.dashboard') }}">Dashboard Writer</a></li>
4 @endif
5 ...

```

Ricordiamo di inserire questa logica all'interno della direttiva `@auth` .

AGGIORNAMENTO DELL'ARTICOLO

Cominciamo ora a gestire la logica dedicata all'**aggiornamento dell'articolo**. Come sempre, partiamo dalla **rotta**, che questa volta sarà parametrica. Andiamo in `web.php` nel gruppo del `writer` :

```

1 Route::middleware('writer')->group(function(){
2     ...
3     Route::get('/article/edit/{article}', [ArticleController::class, 'edit'])->name('article.edit');
4 });

```

Andiamo nell'**ArticleController** nella funzione `edit()`. Per permettere solo al redattore che ha scritto un articolo di poter raggiungere la pagina di modifica, controlliamo se l'id dell'utente loggato è uguale all'id dell'utente che ha scritto l'articolo. Se i due id corrispondono allora l'utente visualizzerà quella pagina, altrimenti lo blocchiamo con un messaggio:

```

1 public function edit(Article $article)
2 {
3     if(Auth::user()->id == $article->user_id){
4         return view('article.edit', compact('article'));
5     }
6     return redirect()->route('homepage')->with('alert', 'Accesso non consentito');
7 }

```

Creiamo quindi adesso un file chiamato `edit.blade.php` all'interno della cartella `resources/views/article/`:

```


1 <x-layout>
2 <div class="container-fluid p-5 bg-secondary-subtle text-center">
3 <div class="row justify-content-center">
4 <div class="col-12">
5 <h1 class="display-1">Modifica l'articolo</h1>
6 </div>
7 </div>
8 </div>
9 <div class="container my-5">
10 <div class="row justify-content-center">
11 <div class="col-12 col-md-8">
12 <form action="#" method="#" class="card p-5 shadow" enctype="multipart/form-data">
13 <div class="mb-3">
14 <label for="title" class="form-label">Titolo</label>
15 <input type="text" name="title" class="form-control" id="title" value="{{article->title}}">
16 @error('title')
17 <span class="text-danger">{{message}}</span>
18 @enderror
19 </div>
20 <div class="mb-3">
21 <label for="subtitle" class="form-label">Sottotitolo</label>
22 <input type="text" name="subtitle" class="form-control" id="subtitle" value="{{article->subtitle}}">
23 @error('subtitle')
24 <span class="text-danger">{{message}}</span>
25 @enderror
26 </div>
27 <div class="mb-3">
28 <label>Immagine Attuale</label>
29 title}}" class="w-50 d-flex">
30 </div>
31 <div class="mb-3">
32 <label for="image" class="form-label">Nuova Immagine</label>
33 <input type="file" name="image" class="form-control" id="image">
34 @error('image')
35 <span class="text-danger">{{message}}</span>
36 @enderror
37 </div>
38 <div class="mb-3">
39 <label for="category" class="form-label">Categoria</label>
40 <select name="category" id="category" class="form-control">
41 <option selected disabled>Seleziona categoria</option>
42 @foreach ($categories as $category)
43 <option value="{{category->id}}" @if(article->category_id == $category->id) selected @endif{{category->name}}</option>
44 @endforeach
45 </select>
46 @error('category')
47 <span class="text-danger">{{message}}</span>
48 @enderror
49 </div>
50 <div class="mb-3">
51 <label for="tags" class="form-label">Tags</label>
52 <input type="text" name="tags" class="form-control" id="tags" value="{{article->tags->implode('name', ', ')}}">
53 <span class="small text-muted fst-italic">Dividi ogni tag con una virgola</span>
54 @error('tags')
55 <span class="text-danger">{{message}}</span>
56 @enderror
57 </div>
58 <div class="mb-3">
59 <label for="body" class="form-label">Corpo del testo</label>
60 <textarea name="body" class="form-control" id="body" cols="30" rows="7">{{article->body}}</textarea>
61 @error('body')
62 <span class="text-danger">{{message}}</span>
63 @enderror
64 </div>
65 <div class="mt-3 d-flex justify-content-center flex-column align-items-center">
66 <button type="submit" class="btn btn-outline-secondary">Modifica articolo</button>
67 <a href="{{route('homepage')}}" class="text-secondary mt-2">Torna alla home</a>
68 </div>
69 </form>
70 </div>
71 </div>
72 </div>
73 </x-layout>

```

Cosa abbiamo fatto di particolare?

1. Abbiamo inserito una sezione per **visualizzare l'immagine attuale** dell'articolo, oltre all'input per poterne inserire una nuova;
2. Nella `<select>`, nel caso ci fosse una categoria collegata all'articolo, la visualizzeremo **già selezionata**;
3. Nell' input dedicato ai **tags**, recuperiamo la collezione dei tags legata all'articolo e poi utilizziamo la funzione `implode()` per trasformare tutti gli elementi che ci interessano (ovvero il *name del tag*) in una stringa.

Fatto questo, andiamo ad aggiornare quindi il tasto all'interno del componente `writer-articles-table.blade.php` inserendo la rotta di modifica:



```
1 <a href="{{route('article.edit', $article)}}" class="btn btn-warning text-white">Modifica</a>
```

Una volta gestita la funzione `edit()` dell'**ArticleController**, dobbiamo spostarci alla funzione `update()` che è dove trova spazio la logica effettiva di aggiornamento dell'articolo all'interno del database. Come sempre, partiamo da `web.php` :



```
1 Route::middleware('writer')->group(function(){  
2     ...  
3     Route::put('/article/update/{article}', [ArticleController::class, 'update'])->name('article.update');  
4 });
```

Spostiamoci quindi nell'**ArticleController**:

```

1 public function update(Request $request, Article $article)
2 {
3     $request->validate([
4         'title' => 'required|min:5|unique:articles,title,' . $article->id,
5         'subtitle' => 'required|min:5',
6         'body' => 'required|min:10',
7         'image' => 'image',
8         'category' => 'required',
9         'tags' => 'required'
10    ]);
11
12    $article->update([
13        'title' => $request->title,
14        'subtitle' => $request->subtitle,
15        'body' => $request->body,
16        'category_id' => $request->category,
17    ]);
18
19    if($request->image){
20        Storage::delete($article->image);
21        $article->update([
22            'image' => $request->file('image')->store('public/images')
23        ]);
24    }
25
26    $tags = explode(',', $request->tags);
27
28    foreach($tags as $i => $tag){
29        $tags[$i] = trim($tag);
30    }
31
32    $newTags = [];
33
34    foreach($tags as $tag){
35        $newTag = Tag::updateOrCreate([
36            'name' => strtolower($tag)
37        ]);
38        $newTags[] = $newTag->id;
39    }
40    $article->tags()->sync($newTags);
41    return redirect(route('writer.dashboard'))->with('message', 'Articolo modificato con successo');
42 }

```

Importiamo:

```

1 use Illuminate\Support\Facades\Storage;

```


Commentiamo un attimo il codice, perché alcuni elementi sono un po' particolari:

- in `$request->validate()`, nel campo `title`, abbiamo fatto un'aggiunta alla regola unique che **ci consente di ignorare l'articolo che stiamo aggiornando**.
 - Abbiamo dovuto farlo perché questa regola controlla se un dato elemento è già presente nel db, e quindi ci avrebbe accettato solo delle modifiche in quel campo che, magari, non vogliamo modificare;
- dopo aver lanciato `explode()` sui tags arrivati dalla request, abbiamo creato un **array di appoggio**.
- Abbiamo poi fatto, nel `foreach`, lo stesso `updateOrCreate()` utilizzato nella funzione di `store()` ma aggiungendo che **gli id dei tag inviati vengano salvati nell'array d'appoggio**.
- gli id salvati in questo array vengono poi passati alla funzione `sync()` che ci **gestisce automaticamente la relazione Many-to-Many tra Article e Tag**: con tutti gli id presenti nell'array, effettuerà un `attach()`, con quelli non presenti effettuerà un `detach()`.
 - Questo ci permetterà di tenere tutte le relazioni ordinate.

Una volta completati questi passaggi, dobbiamo tornare in `article/edit.blade.php` ed aggiungere gli elementi per far funzionare il form:

```
1 <form action="{{route('article.update', $article)}}" method="POST" class="card p-5 shadow" enctype="multipart/form-data">
2     @csrf
3     @method('PUT')
4     ...
```

Abbiamo così gestito correttamente l'aggiornamento dell'articolo da parte del redattore.

CANCELLAZIONE DELL'ARTICOLO 🔗

Occupiamoci adesso, invece, della **cancellazione dell'articolo**. Come sempre, partiamo da `web.php`:

```
1 Route::middleware('writer')->group(function(){
2     ...
3     Route::delete('/article/destroy/{article}', [ArticleController::class, 'destroy'])->name('article.destroy');
4 });
```

Spostiamoci ora nell'**ArticleController**. Prima di cancellare l'articolo, però, dobbiamo prima gestire le relazioni tra l'articolo scelto ed eventuali tags. Nel `foreach` andiamo prima a slegare l'articolo con i suoi tags, per poi cancellarlo:

```

1 public function destroy(Article $article)
2 {
3     foreach ($article->tags as $tag) {
4         $article->tags()->detach($tag);
5     }
6     $article->delete();
7     return redirect()->back()->with('message', 'Articolo cancellato con successo');
8 }

```

Andiamo quindi ad aggiornare il form della cancellazione nel file `writer-articles-table.blade.php`:

```

1 <form action="{{route('article.destroy', $article)}}" method="POST" class="d-inline">
2     @csrf
3     @method('DELETE')
4     <button type="submit" class="btn btn-danger">Elimina</button>
5 </form>

```

Anche nella dashboard del redattore inseriamo il messaggio:

```

1 @if (session('message'))
2     <div class="alert alert-success">
3         {{ session('message') }}
4     </div>
5 @endif

```

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

```

1 git add .
2 git commit -m "User Story 6 completata"
3 git push

```

