

The Aulab Post L11 - EXTRA USER STORY #7

EXTRA User Story #7: [↗](#)

- **Come** Lorenzo
- **vorrei** più informazioni sull'articolo
- **in modo tale** da poter scegliere cosa leggere

ACCEPTANCE CRITERIA: [↗](#)

- slug del titolo nell'uri della pagina di dettaglio
 - calcolo dei minuti di lettura dell'articolo
-

EXTRA - USER STORY 7 - SLUG E TEMPO DI LETTURA [↗](#)

CREAZIONE E GESTIONE SLUG NEL DATABASE [↗](#)

Aggiungiamo ulteriori informazioni ai nostri articoli, per migliorare l'esperienza di navigazione dei nostri utenti. Cominciamo facendo in modo che l'**uri visualizzato nella pagina di dettaglio sia lo slug dell'articolo**. Il primo passo è dunque creare una nuova colonna nel database:

```
1 php artisan make:migration add_slug_to_articles_table
```

Andiamo a modificare la migrazione creata aggiungendo la colonna dello `slug`:

```

1  /**
2   * Run the migrations.
3   */
4  public function up(): void
5  {
6      Schema::table('articles', function (Blueprint $table) {
7          $table->string('slug')->after('title')->nullable()->unique();
8      });
9  }
10
11 /**
12  * Reverse the migrations.
13  */
14 public function down(): void
15 {
16     Schema::table('articles', function (Blueprint $table) {
17         $table->dropColumn('slug');
18     });
19 }

```

Lanciamo quindi la serie di comandi:

```

1 php artisan migrate
2 php artisan migrate:rollback
3 php artisan migrate

```

Andiamo adesso ad aggiungere il nuovo campo nel `fillable` del modello **Article**:

```

1 protected $fillable = [
2     'title', 'subtitle', 'body', 'image', 'user_id', 'category_id', 'is_accepted', 'slug'
3 ];

```

Andiamo ad aggiornare adesso la funzione `store()`, perché sarà **Laravel in automatico a recuperare il titolo dell'articolo e creare lo slug**:

```
1 ...
2 $article = Article::create([
3     'title' => $request->title,
4     'subtitle' => $request->subtitle,
5     'body' => $request->body,
6     'image' => $request->file('image')->store('public/images'),
7     'category_id' => $request->category,
8     'user_id' => Auth::user()->id,
9     'slug' => Str::slug($request->title),
10 ]);
11 ...
```

Importiamo la classe **Str**:

```
1 use Illuminate\Support\Str;
```

Facciamo la stessa cosa nella funzione `update()` sempre nell'**ArticleController**:

```

1  ...
2  $article->update([
3      'title' => $request->title,
4      'subtitle' => $request->subtitle,
5      'body' => $request->body,
6      'category_id' => $request->category,
7      'slug' => Str::slug($request->title),
8  ]);
9  ...

```

Facciamo qualche test per vedere se nel database tutto va come deve. Prima di proseguire, **facciamo in modo che tutti gli articoli all'interno del nostro db abbiano lo slug.**

ROUTE MODEL BINDING E CUSTOMIZZAZIONE DELLA CHIAVE [↗](#)

E' arrivato adesso il momento di utilizzare questo slug all'interno dell'uri.

Il primo passo è andare nel **modello Article** e richiamare questa funzione:

```

1  ...
2  public function getRouteKeyName()
3  {
4      return 'slug';
5  }

```

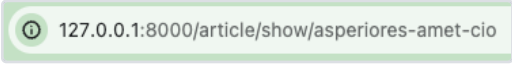
La funzione `getRouteKeyName()` deve necessariamente chiamarsi così.

⚠ Arrivati a questo punto, fai un fresh del database poiché gli articoli creati in precedenza non hanno ancora lo slug. Quindi lancia il comando:

```
1  php artisan migrate:fresh --seed
```

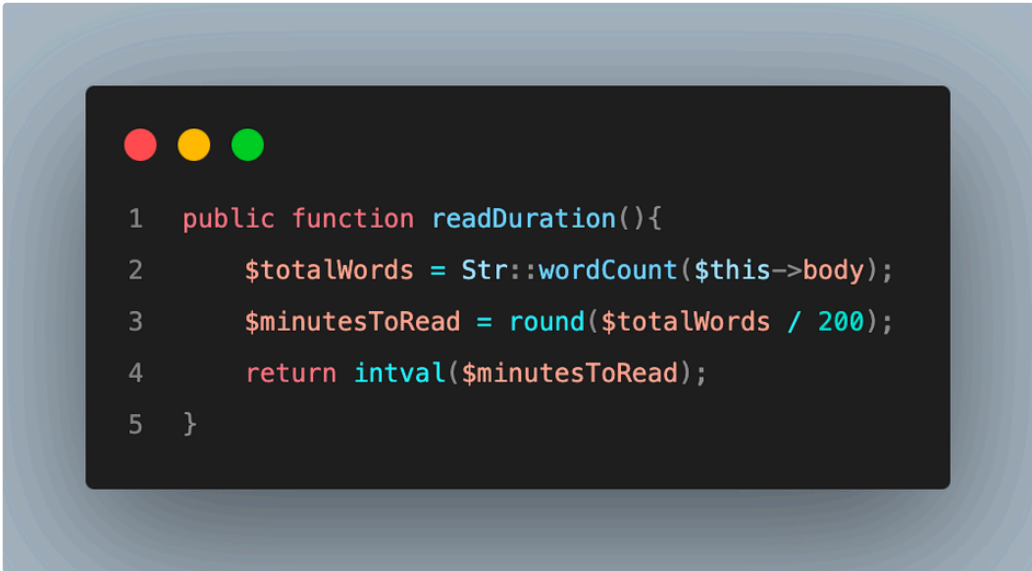
i Con il comando sopra eliminerai automaticamente tutte le tabelle create dalle migrazioni per poi ricrearle; inoltre, il comando eseguirà i seeder del database.

Il risultato nell'URI sarà:




CALCOLARE IL TEMPO DI LETTURA [↗](#)

Il calcolo del tempo di lettura è una feature molto carina ma che recentemente ha trovato un grande utilizzo nel web. Implementarla è molto semplice, e ci basteranno delle semplici funzioni di PHP. Il primo passo è andare nel modello **Article**:



```
1 public function readDuration(){
2     $totalWords = Str::wordCount($this->body);
3     $minutesToRead = round($totalWords / 200);
4     return intval($minutesToRead);
5 }
```

Importiamo:



```
1 use Illuminate\Support\Str;
```

Commentiamo la funzione:

- con `Str::wordCount()` **contiamo il numero delle parole** presenti all'interno del corpo del nostro articolo;
- con `round()` andiamo ad **arrotondare per eccesso** i minuti che ci vogliono per leggere il testo (*la media di una persona scolarizzata è di 200 parole al minuto*). Questo ci restituirà però un dato di tipo `float`;
- è per questo che nel return utilizziamo il metodo `intval()`, che **recupera il valore intero di una data variabile**.

Adesso manca solo visualizzarlo. Possiamo andare in tutte le card ed inserire questo nuovo dato da visualizzare.

Prima di far questo, però, potremmo fare un refactoring del codice rendendo le card un componente. In `resources/views/components` mi creo un file chiamato `article-card.blade.php`.

Ora, vado nella vista `welcome`, mi taglio tutto il codice della card e lo incollo all'interno del componente appena creato. Nel componente `article-card.blade.php` avremo praticamente questo:

```
1 <div class="card" style="width: 18rem;">
2 title }}">
3 <div class="card-body">
4 <h5 class="card-title">{{ $article->title }}</h5>
5 <p class="card-subtitle">{{ $article->subtitle }}</p>
6 @if ($article->category)
7 <p class="small text-muted">Categoria:
8 <a href="{{ route('article.byCategory', $article->category) }}" class="text-capitalize text-muted">
9 {{ $article->category->name }}
10 </a>
11 </p>
12 @else
13 <p class="small text-muted">Nessuna categoria</p>
14 @endif
15 <p class="small text-muted my-0">
16 @foreach ($article->tags as $tag)
17 #{{ $tag->name }}
18 @endforeach
19 </p>
20 <p class="card-subtitle text-muted fst-italic small">tempo di lettura {{ $article->readDuration() }} min</p>
21 </div>
22 <div class="card-footer d-flex justify-content-between align-items-center">
23 <p>Redatto il {{ $article->created_at->format('d/m/Y') }} <br>
24 da <a class="text-muted" href="{{ route('article.byUser', $article->user) }}">{{ $article->user->name }}</a>
25 </p>
26 <a href="{{ route('article.show', $article) }}" class="btn btn-outline-secondary">Leggi</a>
27 </div>
28 </div>
```

In `welcome`, invece, dobbiamo richiamare il componente. Dobbiamo però ricordarci di passare i dati dinamici di ogni articolo. Per farlo, possiamo utilizzare quello che viene chiamato **binding esplicito** (legame esplicito) mettendo il `:` davanti agli attributi HTML. Questo aiuta Laravel a capire che sto passando come dato una variabile e non una stringa statica.

```
1 ...
2 <div class="container my-5">
3 <div class="row justify-content-evenly">
4 @foreach ($articles as $article)
5 <div class="col-12 col-md-3">
6 <x-article-card :article="$article"/>
7 </div>
8 @endforeach
9 </div>
10 </div>
```

Possiamo fare la stessa cosa anche nelle viste `index.blade.php` e `search-index.blade.php`.

Abbiamo completato la settima ed ultima User Story, possiamo pushare:

```
1 git add .  
2 git commit -m "User Story 7 completata"  
3 git push
```

Fine User Story 7

Complimenti! Con questo passaggio abbiamo completato il nostro The Aulab Post!
