Laravel

Passaggi Preliminari

Creazione del progetto:

```
laravel new -n nome_progetto
```

Creazione del database:

```
mysql -uuser -ppassword
create database nome_database
```

Modifica del file di configurazione `.env`

Aggiungere:

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=exam
DB_USERNAME=user
DB_PASSWORD=password
```

Struttura del progetto

Creazione del model e del REST controller

```
php artisan make:model -cmr Genre
```

> **Attenzione:** Il nome della tabella (model) deve essere sempre in inglese e al singolare.

Modifica dei campi della tabella tramite migrazioni

Modificare il file `database/migrations` denominato `create_genres_table` aggiungendo i campi relativi al genere, come ad esempio:

- Nome del genere (stringa di 255 caratteri)

Aggiungiamo quindi:

```
public function up(): void
{
    Schema::create('genres', function (Blueprint $table) {
        $table->id();
        $table->string('name'); // Aggiunto a mano
        $table->timestamps();
    });
}
```

Aggiorniamo quindi le migrazioni. Verrà creata una tabella sul database passato nel file `.env` con i campi appena aggiunti:

```
php artisan migrate
```

Modifica delle route

Modifichiamo il file `web.php` presente in `/routes`, aggiungendo le informazioni relative al controller REST appena creato:

```
<?php

use App\Http\Controllers\GenreController;
use Illuminate\Support\Facades\Route;

Route::resource('/genres', GenreController::class);
```

Sviluppo delle funzionalità CRUD

Read

Modifichiamo il `GenreController` presente in `app/Http/Controllers`, aggiornando il metodo `index`:

```
public function index()
{
    $genre = Genre::all();
    return view('index', compact('genre'));
}
```

Creiamo una view `index.blade.php` in `resources/views` che permetterà di visualizzare i progetti (`project`) che passiamo tramite il controller:

```
<!DOCTYPE html>
<html lang="it">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <title>Progetti</title>
  </head>
  <body>
    <h1><center>Lista dei Generi</center></h1>

    <table border="1px">
      <tr>
        <th>Nome</th>
        <th>Modifica</th>
        <th>Elimina</th>
      </tr>

      @foreach ($genre as $item)
      <tr>
        <td>{{$item->name}}</td>
        <td>Modifica....</td>
        <td>Elimina....</td>
      </tr>
      @endforeach
    </table>
  </body>
</html>
```

Create

Modifichiamo il `GenreController`, aggiornando il metodo `store`:

```
public function store(StoreGenreRequest $request)
{
    $genre = new Genre();
    $genre->name = request('name');
    $genre->save();
    return redirect('/genres');
}
```

Aggiungiamo al file `index.blade.php`:

```
<h3>Inserimento dei progetti</h3>

<form action="/genres" method="post">
  @csrf
  <span>Nome del genere: </span>
  <input type="text" name="name" />
  <input type="submit" value="Invia" />
</form>
```

> **Nota Bene:** L'uso di `@csrf` è fondamentale, altrimenti verrebbe restituito l'errore 419.

Delete

Modifichiamo il `GenreController`, aggiornando il metodo `destroy`:

```
public function destroy(Genre $genre)
{
    $genre->delete();
    return redirect('/genres');
}
```

Aggiungiamo al file `index.blade.php`:

```
<td>
  <form action="/genres/{{ $item->id }}" method="post">
    @csrf @method('DELETE')
    <button type="submit">Elimina</button>
  </form>
</td>
```

Update

Modifichiamo il `GenreController`, aggiornando il metodo `edit`:

```
public function edit(Genre $genre)
{
    return view('edit', compact('genre'));
}
```

Creiamo una view `edit.blade.php` in `resources/views` che permetterà di editare il progetto (`genre`) che passiamo tramite il controller:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <title>Modifica Progetto</title>
  </head>
  <body>
    <h1><center>Modifica Genere</center></h1>

    <form action="/genres/{{ $genre->id }}" method="post">
      @csrf @method('PATCH')
      <span>Nome:</span>
      <input type="text" name="name" value="{{ $genre->name }}" />
      <input type="submit" value="Modifica" />
    </form>
  </body>
</html>
```

Modifichiamo il `GenreController`, aggiornando il metodo `update`:

```
public function update(UpdateProjectRequest $request, Genre $genre)
{
    $genre->name = request('name');
    $genre->save();
    return redirect('/genres');
}
```

Aggiungiamo al file `index.blade.php`:

```
<td>
  <form action="/projects/{{ $item->id }}/edit" method="get">
    <button type="submit">Modifica</button>
  </form>
</td>
```

Test delle funzionalità

Testiamo la funzionalità creata con:

```
php artisan serve
```

Relazioni tra più tabelle

Creiamo ora una nuova entità `Books`, che può essere associata a un genere tramite una chiave esterna. Del libro vogliamo memorizzare i seguenti dati:

- Nome del libro (stringa di 255 caratteri)
- Autore (stringa di 255 caratteri)
- Anno (valore intero)
- Genere (chiave esterna verso la tabella `Genres`)

Creiamo quindi il model e il REST controller con il seguente comando:

```
php artisan make:model -cmr Book
```

Modifichiamo il file presente in `database/migrations`, aggiungendo le informazioni relative alla chiave esterna che collega la tabella `books` alla tabella `genres` con una relazione uno a molti.

```
public function up(): void
{
    Schema::create('books', function (Blueprint $table) {
        $table->id();
        $table->string("name");
        $table->string("author");
        $table->integer("year");
        $table->foreignId("genre_id")
            ->constrained("genres")
            ->onDelete("cascade")
            ->onUpdate("cascade");
        $table->timestamps();
    });
}
```

Il codice crea una colonna `genre_id` come chiave esterna nella tabella `books`, collegata alla colonna `id` della tabella `genres`. Se si utilizza la notazione mostrata nell'esempio, non è necessario specificare manualmente la tabella di riferimento in `constrained`, poiché viene riconosciuta automaticamente.

- `onUpdate('cascade')`: aggiorna automaticamente `genre_id` se l'ID di `genres` cambia.
- `onDelete('cascade')`: elimina i record collegati se un genere viene eliminato.

Modifica delle route

Modifichiamo il file `web.php`, presente in `/routes`, aggiungendo le informazioni relative al controller REST appena creato:

```
<?php

use App\Http\Controllers\GenreController;
use App\Http\Controllers\BookController;
use Illuminate\Support\Facades\Route;

Route::resource('/genres', GenreController::class);
Route::resource('/books', BookController::class); // Aggiunto
```

Modifica dei model

Per poter visualizzare il nome del genere a partire dal libro, è necessario aggiungere una funzione nel model che permetta di restituire l'oggetto `Genre`.

Aggiungiamo i seguenti metodi:

- `belongsTo()`: nella tabella che contiene la chiave esterna (`books`).
- `hasMany()`: nella tabella di riferimento (`genres`).

Modifichiamo quindi il file `Book.php`, presente in `app/Models`, aggiungendo il metodo `belongsTo()`:

```
class Book extends Model
{
    public function genre()
    {
        return $this->belongsTo(Genre::class);
    }
}
```

Modifichiamo il file `Genre.php`, aggiungendo il metodo `hasMany()`:

```
class Genre extends Model
{
    public function books()
    {
        return $this->hasMany(Book::class);
    }
}
```

Ora sarà possibile utilizzare il metodo `genre` per accedere al parametro `name` e visualizzarlo.

Esempio:

```
@foreach ($books as $item)
<tr>
  <td>{{ $item->name }}</td>
  <td>{{ $item->author }}</td>
  <td>{{ $item->year }}</td>
  <!-- Visualizza il nome del genere associato -->
  <td>{{ $item->genre->name }}</td>
</tr>
@endforeach
```

Visualizzazione dell'elenco dei generi nelle fasi di creazione e modifica

Create

Modifichiamo il `BookController`, presente in `app/Http/Controllers`, aggiornando il metodo `index`:

```
public function index()
{
    $books = Book::all();
    $genres = Genre::all();
    return view("books.list", compact("genres", "books"));
}
```

Edit

Modifichiamo il `BookController`, aggiornando il metodo `edit`:

```
public function edit(Book $book)
{
    $genres = Genre::all();
    return view("books.edit", compact("book", "genres"));
}
```

Visualizzazione dell'elenco a cascata

Aggiorniamo i file `index.blade.php` e `edit.blade.php`, aggiungendo il seguente codice:

```
<span>Inserisci genere: </span>
<select name="genre_id">
  @foreach ($genres as $item)
  <option value="{{ $item->id }}" @if ($item->
    id == $book->team_id) selected @endif>{{ $item->name }}
  </option>
  @endforeach
</select>
```

Ora, durante la creazione o la modifica di un libro, sarà possibile selezionare il genere da un elenco a cascata.

Filtrare i libri per genere

Per visualizzare tutti i libri appartenenti a un determinato genere, aggiungiamo al file `web.php`, situato nella cartella `/routes`, il seguente metodo:

```
Route::post('/books/findByGenre', function (Request $request) {
    $books = Book::where('genre_id', request('genre_id'))->get();
    $genres = Genre::all();
    return view("books.list", compact("genres", "books"));
});
```

> **Nota Bene:** È necessario importare Illuminate\Http\Request per il corretto funzionamento.

> **Attenzione:** Tutti i metodi aggiuntivi rispetto a quelli definiti nel controller REST devono essere scritti all'interno del file web.php, sopra le dichiarazioni ::resource() per evitare errori 404.

Nel file `books.list` inseriamo il seguente form:

```
<form action="/books/findByGenre" method="post">
  @csrf
  <span>Inserisci genere: </span>
  <select name="genre_id">
    @foreach ($genres as $item)
    <option value="{{ $item->id }}">{{ $item->name }}</option>
    @endforeach
  </select>
  <button>Filtra</button>
</form>
```

Ordina i libri per anno

Supponiamo di voler ordinare i libri in modo decrescente per anno, aggiungiamo al file `web.php` il seguente metodo:

```
Route::get('/books/orderByYear', function () {
    $books = Books::orderBy('year','desc')->get();
    $genres = Genre::all();
    return view("books.list", compact("genres", "books"));
});
```

Filtrare i libri con anno superiore o uguale a un valore dato

Per visualizzare solo i libri con un anno di pubblicazione pari o superiore a un valore specificato dall'utente, aggiungiamo al file `web.php` il seguente metodo:

```
Route::post('/books/greaterThan', function (Request $request) {
    $books = Books::where('year', '>=', request('year'))->get();
    $genres = Genre::all();
    return view("books.list", compact("genres", "books"));
});
```

Eliminare i libri con anno inferiore ad un valore specificato

Per eliminare tutti i libri con un anno di pubblicazione inferiore a un valore dato dall'utente, aggiungiamo al file `web.php` il seguente metodo:

```
Route::post('/books/lowerThan', function (Request $request) {
    Books::where('year', '<', request('year'))->delete();
    return redirect('/books')
});
```

Dimezzare l'anno di tutti i libri

Per ridurre della metà l'anno di pubblicazione di tutti i libri, aggiungiamo al file `web.php` il seguente metodo:

```
Route::get('/books/halfYear', function () {
    foreach (Book::all() as $book) {
        $book->year = floor($book->year / 2); $book->save();
    }
    return redirect("/books");
});
```

Contare il numero di elementi in una tabella

La funzione `count()` consente di determinare il numero di elementi presenti nella lista passata dal controller. Ad esempio, per visualizzare il numero totale di libri inseriti:

```
<b>Hai inserito: </b>{{ count($books) }}
```

Colorazione dinamica delle righe

Per applicare una colorazione dinamica alle righe della tabella in base all'anno di pubblicazione, aggiungiamo la seguente condizione nel ciclo `foreach` della vista `books.list`:

```
@if ($item->year == 2022)
    <tr style="background-color: coral">
@else
    <tr>
@endif
```

Questo codice evidenzierà in **corallo** tutte le righe corrispondenti a libri pubblicati nel 2022.

Eliminare tutti i record di una tabella

Se vogliamo eliminare tutti i libri presenti nel database, possiamo definire una rotta che esegue l'operazione:

```
Route::get('/books/deleteAll', function () {
    foreach (Book::all() as $item) {
        $item->delete();
    }
    return redirect("/books");
});
```

Questa funzione cicla su tutti i record della tabella `books`, eliminandoli uno per uno, e poi reindirizza l'utente alla lista dei libri.