Pla de Recuperació, Transformació i Resiliència
FPCV Formació Professional Comunitat Valenciana
Unió Europea
Fons social europeu
L'FSE inverteix en el teu futur
GENERALITAT VALENCIANA
Conselleria d'Educació, Cultura, Universitats i Ocupació
IES MARIA ENRIQUEZ

Unitat 5. Frameworks. Laravel

2n DAW - IES María Enríquez

Unitat 5. Frameworks. Laravel

1.Introducció

- 1.1. Frameworks en PHP
- 1.2. Preparació del programari
- 1.3. Primers pasos amb Laravel

2. Rutes i vistes

- 2.1. Rutes amb Laravel
- 2.2. Vistes amb Blade
- 2.3. Estils i javascript

Unitat 5. Frameworks. Laravel

3. Controladors
    3.1. Controladors i tipus
    3.2. Injección de dependències
4. El model de dades
    4.1. Accés a bases de dades
    4.2. Les migracions
    4.3. Els models

Unitat 5. Frameworks. Laravel

5. Model de dades II

    5.1. Relacions entre models

    5.2. Seeders i factories

    5.3. Query Builders i ús de dates

6. Formularis i validació

    6.1. Definició de formularis

    6.2. Validació de dades

6. Formularis i validació

Definició de formularis

Creació i enviament de formularis

La definició d'un formulari en blade és similar que en HTML. Modifiquem la part del `action` passant-li la funció `route.`

Exemple per a donar d'alta un llibre a la biblioteca. Creem la vista `create.blade.php` dins del directori `resources/views/libros`:

Definició de formularis

```
@extends('plantilla')
@section('titulo', 'Nuevo libro')
@section('contenido')
     <h1>Nuevo libro</h1>
     <form action="{{ route('libros.store') }}" method="POST">
     <div class="form-group">
          <label for="titulo">Título:</label>
          <input type="text" class="form-control" name="titulo"
               id="titulo">
     </div>
     <!-- falta input editorial i preu -->
     </div>

     <div class="form-group">
          <label for="autor">Autor:</label>
          <select class="form-control" name="autor" id="autor">
               @foreach ($autores as $autor)
                    <option value="{{ $autor->id }}">
                         {{ $autor->nombre }}
                    </option>
               @endforeach
          </select>
     </div>

     <input type="submit" name="enviar" value="Enviar"
          class="btn btn-dark btn-block">

     </form>
@endsection
```

Definició de formularis

Creació i enviament de formularis

Una segona modificació és que Laravel per defecte protegeix d'atacs **XSS**, per evitar obtenir l'error **419** de formulari no validat. Cal afegir la directiva `@csrf`

```
<form action="{{ route('libros.store') }}" method="POST">
    @csrf
    ...
</form>
```

Definició de formularis

Creació i enviament de formularis

Ara hem de modificar `LibroController`, per a que al tornar la vista `create.blade.resources`, passe el llistat d'autors que apareixeran en el SELECT.

```
use App\Models\Autor;
...
public function create()
{
    $autores = Autor::get();
    return view('libros.create', compact('autores'));
}
```

Definició de formularis

Creació i enviament de formularis

Una vegada enviat el formulari, entrarà en la funció `store` del `LibroController`. Li passarà la informació del formulari mitjançant l'objecte `Request`:

```
public function store(Request $request)
{
    $libro = new Libro();
    $libro->titulo = $request->get('titulo');
    $libro->editorial = $request->get('editorial');
    $libro->precio = $request->get('precio');
    $libro->autor()->associate(Autor::findOrFail($request->get('autor')));
    $libro->save();

    return redirect()->route('libros.index');
}
```

Definició de formularis

Creació i enviament de formularis

Tenim un mètode auxiliar `has` per controlar si existeix un camp:

```
public function store(Request $request)
{
    if($request->has('titulo'))
    {
        ...
    }
}
```

Definició de formularis

Creació i enviament de formularis

Per últim caldrà afegir al menú de navegació (`views/partials/nav.blade.php`) l'enllaç per afegir un llibre.

```
<nav class="navbar navbar-expand-lg navbar-dark bg-secondary">
    ...
    <div class="collapse navbar-collapse" id="navbarNav">
        <ul class="navbar-nav">
            ...
            <li class="{{ setActivo('libros.create') }} nav-item">
                <a class="nav-link" href="{{ route('libros.create') }}">
                Nuevo libro</a>
            </li>
        </ul>
    </div>
</nav>
```

Definició de formularis

Actualitzacions i borrat

Per defecte els formularis utilitzen els mètodes `POST` i `GET`. Si volem modificar o eliminar el formulari ha d'anar associat al mètode `PUT` o `DELETE`.

Per fer-ho, utilitzarem la directiva `@method`

```
<form ...>
    @csrf
    @method('PUT')
    ...
</form>
```

Definició de formularis

Actualitzacions i borrat

En la fitxa del llibre (`views/libros/show.blade.php`), podem afegir un formulari per eliminar:

```
<form action="{{ route('libros.destroy', $libro->id) }}" method="POST">
    @csrf
    @method('DELETE')
    <input type="submit" class="btn btn-danger" value="Borrar libro" />
</form>
```

Definició de formularis

Actualitzacions i borrat

Com li passem la `id` en la ruta, ara només caldrà, des de `LibroControllador`, el mètode `destroy`, farà el delete:

```
public function destroy($id)
{
    $libro = Libro::findOrFail($id);
    $libro->delete();
    return redirect()->route('libros.index');
}
```

Validació de dades

Per fer la validació de dades utilitzarem l'objecte request que ens proporciona el mètode validate al qual li passarem les regles de validació.

```
public function store()
{
    request()->validate(
        [
            'titulo' => 'required|min:3',
            'editorial' => 'required',
            'precio' => 'required|numeric|min:0'
        ]
    );

    // ... Codi per processar la petició
}
```

Validació de dades

Ús de form request

Si les regles de validació són més complexes, caldrà crear un form request, que és una classe addicional que contindrà la lògica de validació.

```
php artisan make:request LibroPost
```

Aquesta classe es crea en `app/Http/Requests` amb els mètodes:

- `authorize`: s'utilitza per controlar permisos. De moment farem que retorne true.
- `rules`: retorna un array de validacions.

Validació de dades

```
public function rules()
{
    return [
        'titulo' => 'required|min:3',
        'editorial' => 'required',
        'precio' => 'required|numeric|min:0'
    ];
}
```

Posteriorment, injectem el form request en el store de LibroController. La validació és automàtica:

```
public function store(LibroPost $request)
{
    // Si entramos aquí, el formulario es válido
}
```

Validació de dades

Mostrar missatge d'error

Si la validació falla, retornarà a la pàgina del formulari amb la informació de l'error que està a la variable `$errors`. Aquest variable té un mètode `any`, que indica si hi ha cap error. També el mètode `all` que retorna un array amb tots els errors.

```
@if ($errors->any())
    <ul>
    @foreach($errors->all() as $error)
        <li>{{ $error }}</li>
    @endforeach
    </ul>
@endif
<form ...>
    @csrf
    ...
</form>
```

Validació de dades

Mostrar missatge d'error

També podem mostrar els errors per camp amb el mètode `first`

```
<form action="{{ route('libros.store') }}" method="POST">
   @csrf
   <div class="form-group">
      <label for="titulo">Título:</label>
      <input type="text" class="form-control" name="titulo"
         id="titulo">
      @if ($errors->has('titulo'))
         <div class="text-danger">
            {{ $errors->first('titulo') }}
         </div>
      @endif
   </div>
```

Validació de dades

Mostrar missatge d'error

Podem personalitzar els missatges d'error modificant el mètode `messages` del form request:

```
public function messages()
{
    return [
        'titulo.required' => 'El título es obligatorio',
        ...
    ];
}
```

Validació de dades

Mostrar missatge d'error

Si validem les dades en el controlador, podem passar-ho com a segon paràmetre quan cridem al mètode `validate`:

```
request()->validate(
    [
        'titulo' => 'required|min:3',
        'editorial' => 'required',
        'precio' => 'required|numeric|min:0'
    ], [
        'titulo.required' => 'El título es obligatorio',
        ...
    ]
);
```

Validació de dades

Recordar valors enviats

Si volem recuperar les dades enviades en un formulari utilitzem el mètode `old`:

```
<form action="{{ route('libros.store') }}" method="POST">
    @csrf

    <div class="form-group">
        <label for="titulo">Título:</label>
        <input type="text" class="form-control" name="titulo"
            id="titulo" value="{{ old('titulo') }}">
        @if ($errors->has('titulo'))
            <div class="text-danger">
                {{ $errors->first('titulo') }}
            </div>
        @endif
    </div>
```

Exercicis

Exercici 1

Crea un formulari per a donar d'alta nous posts, en la vista `resources/views/posts/create.blade.php`. Afig un parell de camps (un text curt i un text llarg) per a emplenar el títol i el contingut, i com a autor o usuari del post de moment deixa un predefinit; per exemple, l'autor amb id = 1, o el primer usuari que trobes en la base de dades (`Usuari::get()->first()`). Més endavant ja ho farem dependent de l'usuari que s'haja autenticat. Recorda definir el mètode `store` en el controlador de posts per a donar d'alta el post, i redirigir després al llistat principal de posts. Per a carregar el formulari, afig una nova opció en el menú principal de navegació.

Exercicis

Exercici 2

Ara afegirem el formulari d'edició d'un post, també des de la vista de la fitxa del post. El formulari haurà de mostrar les dades ja farcides del post. Aquest formulari es carrega a partir del mètode `edit` (que haurà de renderitzar la vista amb el formulari d'edició, `resources/views/posts/edit.blade.php`), i el formulari s'enviarà al mètode `update` del controlador, passant-li com a paràmetre l'`id` del post a modificar.

Exercicis

Exercici 3

Crea un form request anomenat `PostRequest`, que valide les dades del post. En concret, han de complir-se aquests requisits:

- El títol del post ha de ser obligatori, i d'almenys 5 caràcters de longitud
- El contingut del post ha de ser obligatori, i d'almenys 50 caràcters de longitud

Defineix missatges d'error personalitzats per a cada possible error de validació, i mostra'ls al costat de cada camp afectat, com en l'exemple de la biblioteca. A més, utilitza la funció `old` per a recordar el valor antic correcte, en el cas que un camp passe la validació però un altre(s) no.

Atribucions

Curs de Nacho Iborra