Laravel

Practical 4

James Ooi

Mar 2018

Contents

1 Validation

Let's implement validation for our **store** method of **MemberController**.

1.1 The Validation Rules

Assume that the input data for members must conform to the following rules:

- **membership_no**: Required, unique, has the format **XXX9999999**
- **nric**: Required, has the format **YYMMDD999999**
- **name**: Required, maximum 100 characters
- **address**: Required, maximum 500 characters
- **postcode**: Required, has the format **99999**
- **city**: Required, maximum 50 characters
- **state**: Required
- **phone**: Required, has the format **999-99999999** where the area code has 2 or 3 digits while the number has 6 to 8 digits
- **division_id**: Required

To achieve the above validation rules, we will be using the following built-in Laravel

- **required**
- **unique**
- **max**
- **regex**

Modify the **store** method of **MemberController** as follows:

```
1  /**
2   * Store a newly created resource in storage.
3   *
4   * @param  \Illuminate\Http\Request  $request
5   *
6   * @return \Illuminate\Http\Response
7   */
```

```
8  public function store(Request $request)
9  {
10     $request->validate([
11         'membership_no' => [
12             'required',
13             'unique:members',
14             'regex:/^([A-Z]{3})([0-9]{7})$/'
15         ],
16         'nric' => [
17             'required',
18             'regex:/^([0-9]{2})([0-1]{1})([0-9]{1})([0-3]{1})([0-9]{1})([0-9]{6})$/',
19         ],
20         'name' => 'required|max:100',
21         'address' => 'required|max:500',
22         'postcode' => [
23             'required',
24             'regex:/^([0-9]{5})$/',
25         ],
26         'city' => 'required|max:50',
27         'state' => 'required',
28         'phone' => [
29             'required',
30             'regex:/^([0-9]{2,3})\-([0-9]{6,8})$/',
31         ],
32         'division_id' => 'required',
33     ]);
34
35     $member = new Member;
36     $member->fill($request->all());
37     $member->save();
38
39     return redirect()->route('member.index');
40 }
```

1.2 Displaying Error Messages

Add the following code before the HTML Form in the **create.blade.php** view file:

```
1 @if ($errors->any())
2   <div class="alert␣alert-danger">
3     <ul>
4       @foreach ($errors->all() as $error)
5         <li>{{ $error }}</li>
6       @endforeach
7     </ul>
8   </div>
9 @endif
```

1.3 Testing Validation

Test your implementation with various data, both valid and invalid data to verify that the implemented valiation rules works exactly as required.

1.4 Tasks (Unguided)

- Implement validation for the **update** method of **MemberController**.
- Define appropriate validation rules for each attribute in the **Division** model and implement validation for the **store** and **update** methods of **DivisionController**.
- Define appropriate validation rules for each attribute in the **Group** model and implement validation for the **store** and **update** methods of **GroupController**.
- Redo all the above using *form requests*.

2 Custom Validation Rules

In this section, we shall create custom validation rules for some of the attributes in **Member**. Remember that we have used regular expression to validate attributes such as **membership_no**, **nric**, **postcode** and **phone**. We shall define a custom validation rule for each of these attributes to implement validation.

2.1 Defining the Rule

Issue the following Artisan CLI command to create a rule named **MembershipNo**:

```
php artisan make:rule MembershipNo
```

The **MembershipNo** class file will be generated in the directory **app/Rules**. We shall now define our validation rules in the **MembershipNo** rule class file.

Define the **passes** method as follows:

```
/**
 * Determine if the validation rule passes.
 *
 * @param  string  $attribute
 * @param  mixed  $value
 * @return bool
 */
public function passes($attribute, $value)
{
    return preg_match('/^([A-Z]{3})([0-9]{7})$/', $value);
}
```

Define the **message** method as follows:

```
/**
 * Get the validation error message.
 *
 * @return string
 */
public function message()
{
    return 'The :attribute is invalid.';
}
```

Apply the custom rule in your validation rules. Assuming you have completed the previous tasks, you should be implementing this in your corresponding form request class. Otherwise, you can also implement it in the **validate** method in your controller:

```
'membership_no' => [
    'required',
```

```
3       'unique:members',
4       new MembershipNo,
5   ],
```

2.2 Tasks (Unguided)

- Implement custom validation rules for the **nric**, **postcode** and **phone** attributes of **Member** and apply them for both the **store** and **update** methods of the **MemberController**.
- Implement custom validation rules for all attributes in **Division** that uses regular expression validation and apply them for both the **store** and **update** methods of the **DivisionController**.
- Implement custom validation rules for all attributes in **Group** that uses regular expression validation and apply them for both the **store** and **update** methods of the **GroupController**.

3 Uploading Files

In this section, we will implement functionalities to allow a staff to upload a photo for each member.

3.1 The Routes

```
1  Route::get('/member/{member}/upload', 'MemberController@upload')
2      ->name('member.upload');
3  Route::post('/member/{member}/save-upload', 'MemberController@saveUpload')
4      ->name('member.saveUpload');
```

3.2 The Upload Form

3.2.1 The *upload* Method

```
1  /**
2   * Show the form for uploading a photo.
3   *
4   * @return \Illuminate\Http\Response
5   */
6  public function upload($id)
7  {
8      $member = Member::find($id);
9      if(!$member) throw new ModelNotFoundException;
10
11     return view('members.upload', [
12         'member'     => $member,
13     ]);
14 }
```

3.2.2 The *upload.blade.php* View

```
1  @extends('layouts.app')
2
3  @section('content')
4
5      <!-- Bootstrap Boilerplate... -->
6
7      <h3>Upload Photo</h3>
8      <h4>Membership No.: <em>{{ $member->membership_no }}</em></h4>
9      <h4>Member Name: <em>{{ $member->name }}</em></h4>
10
11     <div class="panel-body">
12         @if ($errors->any())
13             <div class="alert␣alert-danger">
```

```
                <ul>
                    @foreach ($errors->all() as $error)
                        <li>{{ $error }}</li>
                    @endforeach
                </ul>
            </div>
        @endif

        <!-- Upload Form -->
        {!! Form::open([
            'route' => ['member.saveUpload', $member->id],
            'class' => 'form-horizontal',
            'enctype' => 'multipart/form-data',
        ]) !!}

            <!-- Code -->
            <div class="form-group row">
                {!! Form::label('member-photo', 'Select File', [
                    'class'         => 'control-label col-sm-3',
                ]) !!}
                <div class="col-sm-9">
                    {!! Form::file('image', [
                        'id'        => 'member-photo-file',
                        'class'     => 'form-control',
                    ]) !!}
                </div>
            </div>

            <!-- Submit Button -->
            <div class="form-group row">
                <div class="col-sm-offset-3 col-sm-6">
                    {!! Form::button('Upload', [
                        'type'          => 'submit',
                        'class'         => 'btn btn-primary',
                    ]) !!}
                </div>
            </div>
        {!! Form::close() !!}
    </div>

@endsection
```

3.2.3 Link to Upload Form

Modify the **index.blade.php** view file to include a link to open the upload form for each member. Add a link beside the existing **Edit** link in each row:

```
{!! link_to_route(
    'member.upload',
    $title = 'Upload Photo',
```

```
    $parameters = [
        'id' => $member->id,
    ]
) !!}
```

3.3 Validation

Create a form request named **UploadPhotoRequest**:

```
namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class UploadPhotoRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return true;
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules()
    {
        return [
            'image' => 'required|image|mimes:jpeg|max:2000'
        ];
    }
}
```

In the above form request, we specify in the validation rules that the **image** attribute must be a JPEG image file with a maximum file size of 2,000 kb.

3.4 Handling File Upload

Now, we complete the code for the **saveUpload** method to handle the file upload:

```
/**
```

```
 2   * Store a newly created resource in storage.
 3   *
 4   * @param  \Illuminate\Http\Request  $request
 5   *
 6   * @return \Illuminate\Http\Response
 7   */
 8  public function saveUpload(UploadPhotoRequest $request, $id)
 9  {
10      $file = $request->file('image');
11
12      $path = $file->storeAs('public/members', $id.'.jpg');
13
14      return redirect()->route('member.index');
15  }
```

In the above code, we store the uploaded file using the to the path **public/members** in the **app/storage/app** directory.

3.5 Linking Public Accessible Directory to Storage

In the **app/storage/app** directory, the **public** is intended for files that are going to be publicly accessible. To make them accessible from the web, you should create a symbolic link from **public/storage** to **storage/app/public**.

To create the symbolic link, you may use the storage:link Artisan command:

```
php artisan storage:link
```

3.6 Displaying the Photo

We now modify the **show.blade.php** view to display the uploaded member's photo.

```
1  @if(Storage::disk('public')->exists('members/'.$member->id.'.jpg'))
2      <img src="/storage/members/{{$member->id}}.jpg"
3        width="240" alt="{{ $member->membership_no }}">
4  @endif
```