Department of Information Technology
IT 350 - Web Technologies
Fall - 2021

Databases and Eloquent III

Lecture Eight

Rebin M. Ahmed
rebin.mohammed@tiu.edu.iq

Previous Lecture

- Overview
- Eloquent Operations
- Defining Migrations
- Running Migrations
- More Examples

Contents

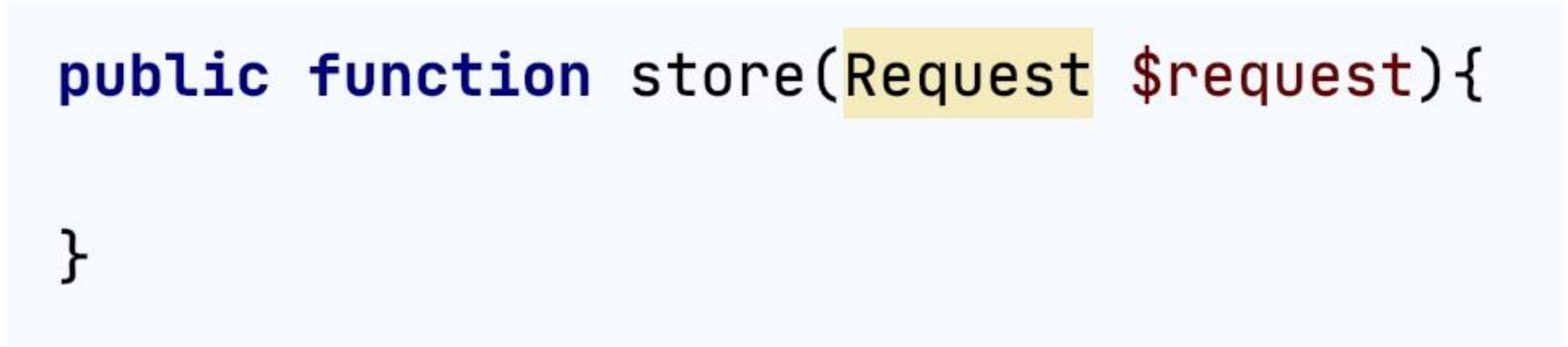
Collecting and Handling User Data

Websites that benefit from a framework like Laravel often don't just serve static content. Many deal with complex and mixed data sources, and one of the most common (and most complex) of these sources is user input in its myriad forms: URL paths, query parameters, POST data, and file uploads.

Laravel provides a collection of tools for gathering, validating, normalizing, and filtering user-provided data. We'll look at those here.

Collecting and Handling User Data

The most common tool for accessing user data in Laravel is injecting an instance of the *Illuminate\Http\Request* object. It provides easy access to all of the ways users can provide input to your site: POST, posted JSON, GET (query parameters), and URL segments.



```php
public function store(Request $request){

}
```

Collecting and Handling User Data

There's also a ***request()*** global helper and a Request facade, both of which expose the same methods. Each of these options exposes the entire Illuminate Request object, but for now we're only going to cover the methods that specifically relate to user data.

Collecting and Handling User Data

$request->all()

Just like the name suggests, $request->all() gives you an array containing all of the input the user has provided, from every source. Let’s say, for some reason, you decided to have ...

Collecting and Handling User Data

$request->except() and ***$request->only()***

$request->except() provides the same output as $request->all(), but you can choose one or more fields to exclude—for example, _token. You can pass it either a string or an array of strings.

$request->only() is the inverse of ***$request->except()***, you will only get the field that you specify in the ***only()*** method.

Collecting and Handling User Data

$request->has()

With ***$request->has()*** you can detect whether a particular piece of user input is available to you.

$request->method()

returns the HTTP verb for the request, and ***$request- >isMethod()*** checks whether it matches the specified verb

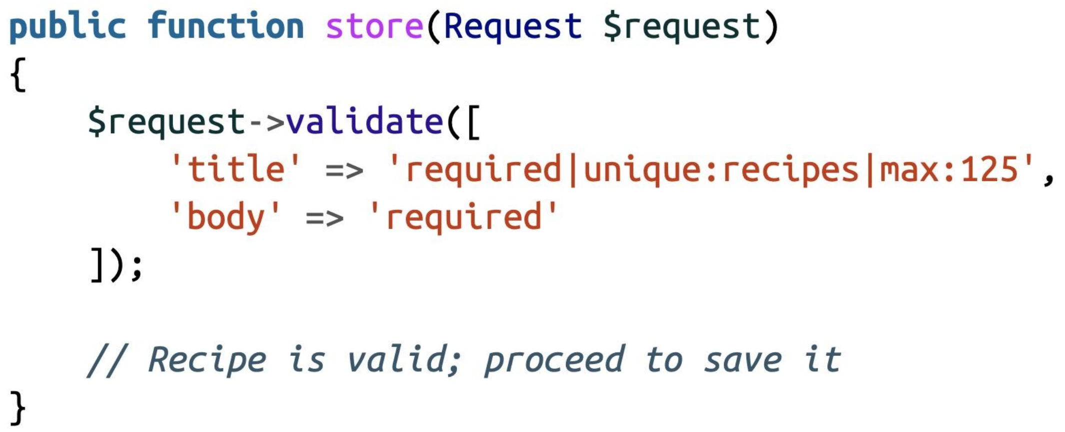
Validation

Laravel has quite a few ways you can validate incoming data. We'll cover form requests in the next section, so that leaves us with two primary options: validating manually or using the ***validate()*** method on the Request object. Let's start with the simpler, and more common, validate().

Validation

validate() on the Request Object

The Request object has a ***validate()*** method that provides a convenient shortcut for the most common validation workflow.



```
public function store(Request $request)
{
    $request->validate([
        'title' => 'required|unique:recipes|max:125',
        'body' => 'required'
    ]);

    // Recipe is valid; proceed to save it
}
```

Validation

We only have four lines of code running our validation here, but they’re doing a lot.

First, we explicitly define the fields we expect and apply rules **(here separated by the pipe character, |)** to each individually.

Next, the validate() method checks the incoming data from the $request and determines whether or not it is valid.

If the data is valid, the validate() method ends and we can move on with the controller method, saving the data or whatever else.

Validation

But if the data isn't valid, it throws a ValidationException. This contains instructions to the router about how to handle this exception.

In our examples here we're using the **"pipe"**

syntax: 'fieldname': 'rule|otherRule|anotherRule'.

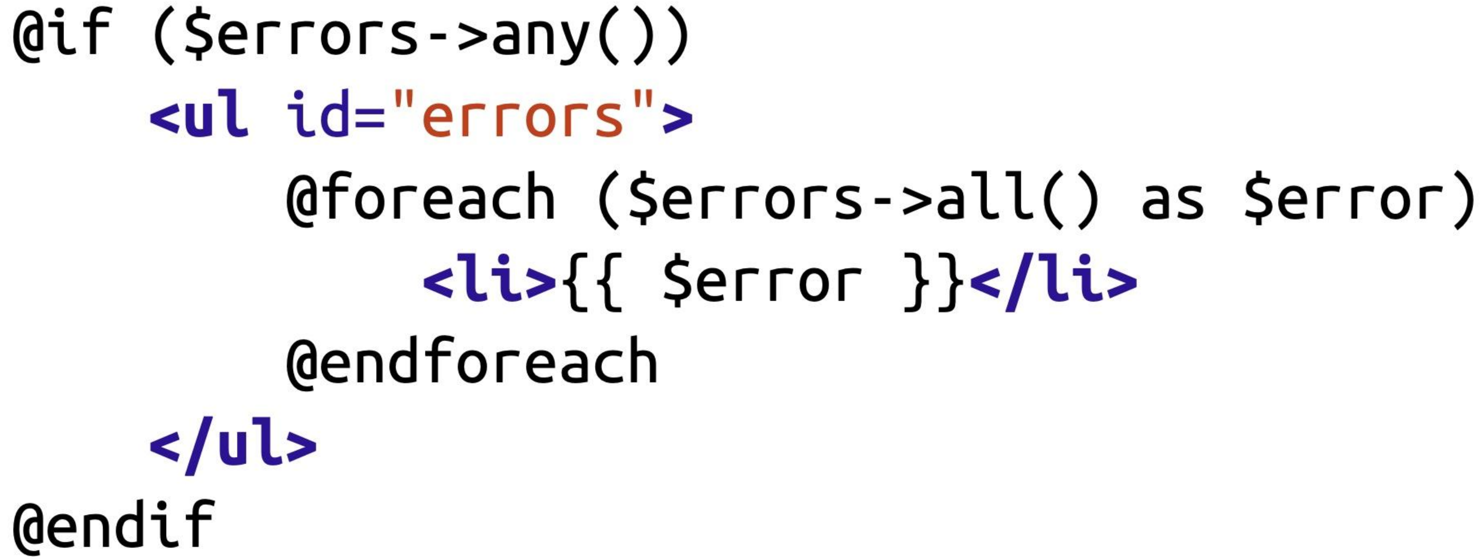
But you can also use the array syntax to do the same thing:

'fieldname': ['rule' , 'otherRule' , 'anotherRule'].

Validation

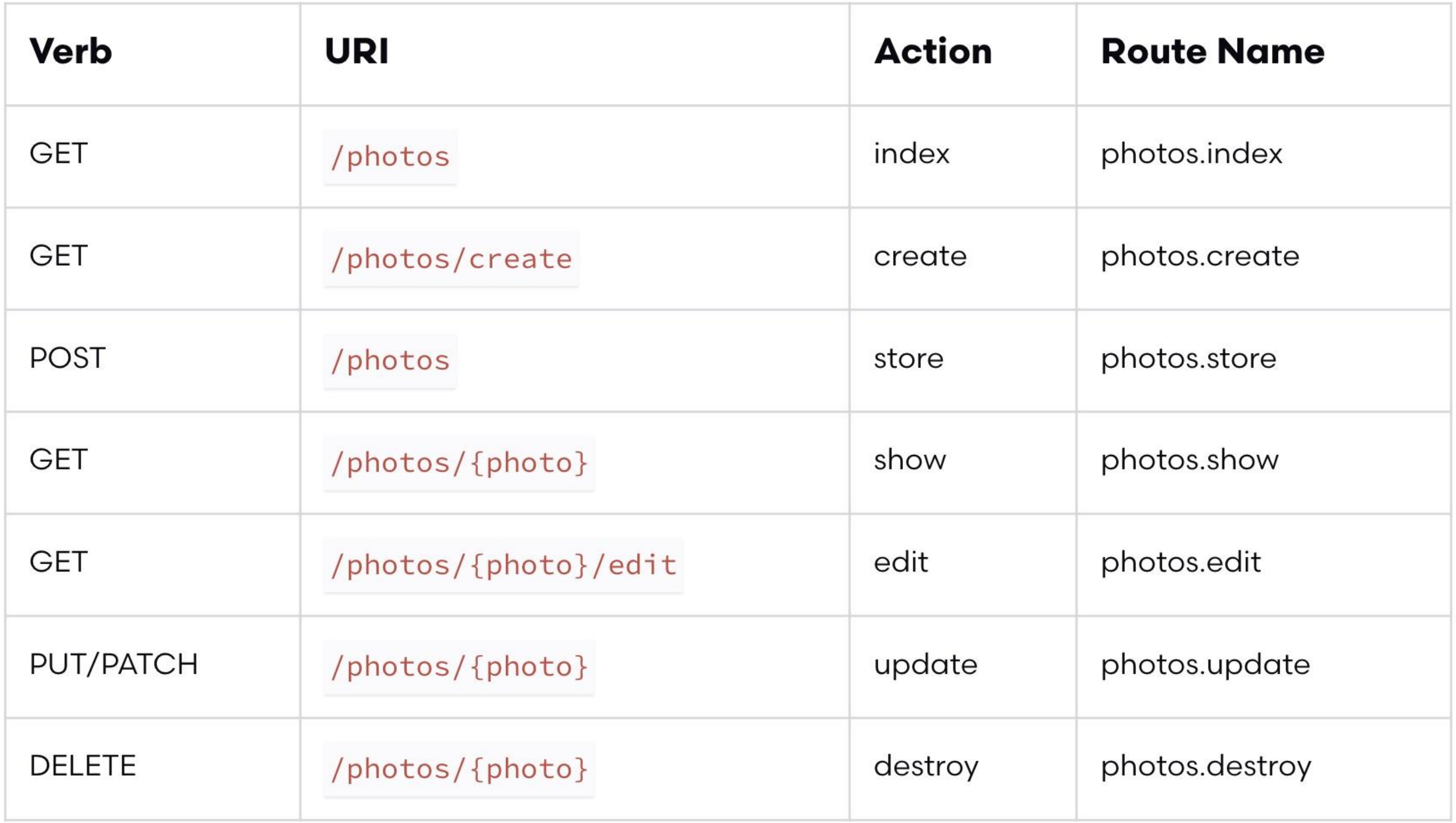
The ***validate()*** method on requests (and the withErrors() method on redirects that it relies on) flashes any errors to the session. These errors are made available to the view you're being redirected to in the ***$errors*** variable. And remember that as a part of Laravel's magic, that ***$errors*** variable will be available every time you load the view, even if it's just empty, so you don't have to check if it exists with ***isset()***.

Validation



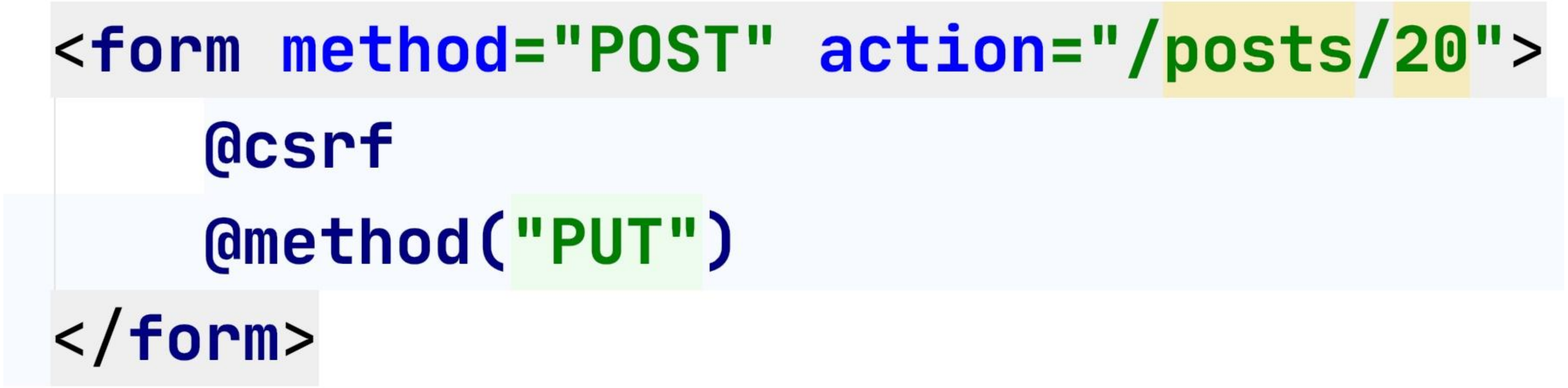
```
@if ($errors->any())
    <ul id="errors">
        @foreach ($errors->all() as $error)
            <li>{{ $error }}</li>
        @endforeach
    </ul>
@endif
```

REST Routes and Actions



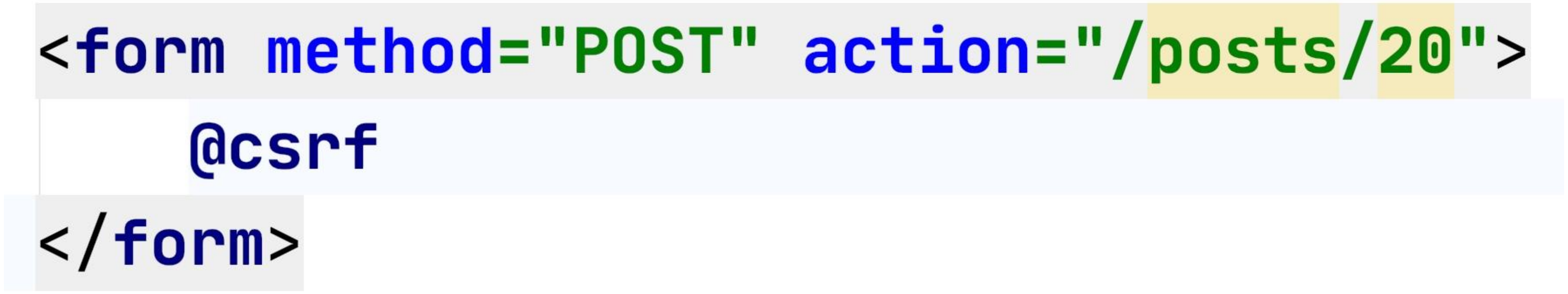
Verb	URI	Action	Route Name
GET	`/photos`	index	photos.index
GET	`/photos/create`	create	photos.create
POST	`/photos`	store	photos.store
GET	`/photos/{photo}`	show	photos.show
GET	`/photos/{photo}/edit`	edit	photos.edit
PUT/PATCH	`/photos/{photo}`	update	photos.update
DELETE	`/photos/{photo}`	destroy	photos.destroy

Update form



```
<form method="POST" action="/posts/20">
    @csrf
    @method("PUT")
</form>
```

Delete Form



```
<form method="POST" action="/posts/20">
    @csrf
</form>
```

Activities and Next Week Topics

This Week:

- Practise the different types of Validation.
- Practise Update and Delete with a different examples.
- Apply Validation, Update, and Delete techniques to your projects.

Next Week:

- User Authentication and Authorization(Register, Login, Reset Password, Forget Password, Email Verification)

References / Further Readings

- Laravel.com : Laravel’s official Documentation.
- Matt Stauffer, 2019. Laravel: Up & Running: A Framework for Building Modern PHP Apps. O’Reilly Media.
- Dayle Rees, 2016. Laravel: Code Smart.