



# WEB разработка

## Laravel Requests

# Съдържание

- Requests
- Валидация на данни в контролера
- Custom Requests
- Error messages /съобщения след неуспешна валидация на формите/
- Custom Error Messages
- Laravel Session - success messages /съобщения след успешна валидация на формите/

# Задача 1

1. Направете валидация на данните постъпващи от потребителите във всички форми на вашето приложение /за създаване и промяна на данните/.

\*\*\*\*

2. Изведете съобщения за грешка/ки при валидация.

\*\*\*\*

3. Изведете съобщения за успешно създаване/промяна/изтриване на данни.

**Requests**

# Requests

## Достъп до данните, прехвърляни със текущата заявка

Имаме директен достъп до инстанция на текущата заявка - данните, които се прехвърлят с нея.

За целта като параметър на метода посочваме `$request`, който трябва да бъде инстанция на `Illuminate\Http\Request` клас /т.нар. `dependency injection` и `type-hint`/.

# Requests

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class UserController extends Controller
{
    /**
     * Store a new user.
     *
     * @param Request $request
     * @return Response
     */
    public function store(Request $request)
    {
        $name = $request->input('name');

        //
    }
}
```

# Requests

## Dependency Injection & Route Parameters

Ако методът в контролера очаква данни посредством параметър, подаван с route, изброявате тези параметри след другите зависимости

Ако пътят ви е дефиниран като

```
Route::put('user/{id}', 'UserController@update');
```

В контролера и съответния метод ги посочвате -

```
public function update(Request $request, $id)
{
    //
}
```

# Requests

## Достъп до URL на заявката

Пълния URL на постъпващата заявка достъпваме с методите `url` или `fullUrl`.

```
// Without Query String...
```

```
$url = $request->url();
```

```
// With Query String...
```

```
$url = $request->fullUrl();
```



# Requests

Достъп до метода на постъпващата заявка

```
$method = $request->method();
```

```
if ($request->isMethod('post')) {  
    //  
}
```

# Requests

## Достъп до uri на текущата заявка

**path()** - връща информация за пътя/uri на текущата заявка.

Ако постъпващата заявка е към <http://domain.com/foo/bar>, методът ще върне **foo/bar**:

```
$uri = $request->path();
```

Предназначение - напр. Проверка дали пътя съответства на даден шаблон, за добавяне на клас, допълнителна логика ...

**is()** методът прави проверката.

Методът ви дава възможност да проверите дали пътя на постъпващата заявка отговаря на даден шаблон.

Може да използвате и **\*** в построяването на шаблона.

```
if ($request->is('admin/*'))  
{  
    //  
}
```

# Requests

## Достъп до въведените от потребителя данни

Всички данни - методът връща данните под формата на масив

```
$input = $request->all();
```

Достъп до въведена стойност

```
$name = $request->input('name');//връща стойността, въведена в полето 'name'
```

# Requests

Достъп до част от въведените данни - `only` / `except`

```
$input = $request ->only(['username', 'password']);
```

```
$input = $request->only('username', 'password');
```

```
$input = $request->except(['credit_card']);
```

```
$input = $request->except('credit_card');
```

# Requests

## Определяне дали съществува въведена стойност - has()

Методът връща true ако има въведена стойност и тя не е празен стринг.

```
        if ($request->has('name')) {  
            //  
        }
```

Ако е подаден масив - методът ще провери дали всички негови стойности отговарят на горното условие

```
        if ($request->has(['name', 'email'])) {  
            //  
        }
```

# Requests

## **flash()**

Методът запазва въведените данни в сесията, с цел да бъдат използвани при следващата заявка на потребителя /след това се изтриват/

```
$request->flash();
```

С **flashOnly()** и **flashExcept()** запазвате част от данните в сесията.

С тях можете да изключите например паролата от запазване в сесията.

```
$request->flashOnly(['username', 'email']);
```

```
$request->flashExcept('password');
```

# Requests

Запазване на данните до следващата заявка и пренасочване - `redirect()->withInput()`

```
return redirect('form')->withInput(  
    $request->except('password')  
);
```

# Requests

## Извеждане на данни с old() /Old Input/

Методът извежда данните от предишната заявка, запазени в сесията.

```
$username = $request->old('username');
```

Ако трябва да изведете данни /например при неуспешна валидация/ в [Blade](#) шаблоните, може да използвате old() методът.

Ако не са въведени данни /няма 'стари' данни/, методът връща null.

```
<input type="text" name="username" value="{{ old('username') }}">
```

[Допълнително инфо](#)



Валидация на данни в контролера

# Валидация на данни в контролера

Ако разгледате базовия controller (App\Http\Controllers\Controller) class, ще видите, че той използва ValidatesRequests trait.

**ValidatesRequests** предоставя метод за валидация, достъпен за всички контролери.

Методът validate() приема постъпващата HTTP заявка и набор от правила за валидация.

Ако условията за валидация са изпълнени, ще продължи изпълнението на следващия код.

Ако валидацията е неуспешна - Ларавел връща съответната грешка към потребителя.

При HTTP заявка получаваме пренасочване /обратно, ако не е посочено друго/

# Валидация на данни в контролера

```
/**
 * Store a new lecture.
 *
 * @param Request $request
 * @return Response
 */
public function store(Request $request)
{
    $this->validate($request, [
        'title' => 'required|unique:posts|max:255',
        'body' => 'required',
    ]);

    // The lecture is valid, store in database...
}
```

←--

Подаваме постъпващата HTTP заявка и правилата за валидация като параметри на `validate` метода.

Ако валидацията е неуспешна - автоматично се генерира подходящия отговор - пренасочване назад.

Ако валидацията е успешна - продължава изпълнението на следващия код.

# Custom Requests

# Custom Requests

## Създаване Form Requests

За по-сложни случаи на валидация създаваме "form request".

Това са request класове, съдържащи валидационна логика.

Създаваме ги с командата

**php artisan make:request StoreLecture**

Новосъздадения клас се намира в [app/Http/Requests](#).  
/ако директорията не съществува - тя се създава след изпълнението на командата/

Методът **rules()** съдържа правилата за валидация

```
/**
 * Get the validation rules that apply to the request.
 *
 * @return array
 */

public function rules()
{
    return [
        'title' => 'required|unique:posts|max:255',
        'body' => 'required',
    ];
}
```

# Custom Requests

След създаването на custom request -

**StoreLecture**, в метода на контролера като параметър подаваме /type-hint/ от какъв клас е \$request.

Постъпващата заявка се валидира преди да бъде изпълнен метода от контролера и

**не е нужно да се добавя друга валидационна логика в него!**

При неуспешна валидация се потребителя се пренасочва обратно.

Грешките от валидацията се записват в сесията до следващата заявка и могат да бъдат изведени като съобщение в браузъра.

```
/**
 * Store the incoming lecture.
 *
 * @param StoreLectureRequest $request
 * @return Response
 */
public function store(StoreLectureRequest
$request)

{
    // The incoming request is valid...
}
```

# Custom Requests

## Authorizing Form Requests

*/\*изпреварваме с тази информация, тъй като още не сме говорили за проверка на потребителите/*

Request класът, създаден от нас съдържа метод `authorize`. В този метод, служи за проверка дали потребителят има права да изпълнява методът от контролера, към който е свързан `request` класа. Например - правим проверка дали логнатия потребител има права да добавя нови лекции.

```
/**
 * Determine if the user is authorized to make this request.
 *
 * @return bool
 */

public function authorize()
{
    $lecture = Lecture::find($this->route('lecture'));

    return $lecture && $this->user()->can('create', $lecture);
}
```

# Custom Requests

Създаваните от нас request класове наследяват базовия Laravel request class и имаме възможност да използваме user метода, с който достъпваме логнатия потребител.

С route методът достъпваме всички URI параметри, дефинирани за пътя, който извикваме.

Като например {lecture} параметъра /в случая, когато искаме да променим лекция/

```
Route::post('lecture/{lecture}');
```

Когато методът authorize върне false, получавате HTTP отговор с код 403 /нямате права да извършвате заявката/ и методът в контролера няма да бъде изпълнен.



# Custom Requests

Когато планирате логиката за проверка на потребителя да бъде в друга част от приложението, методът `authorize` трябва да връща **true**:

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

# Правила за валидация в Ларавел

# Правила за валидация в Ларавел

```
$this->validate($request, [
```

```
    'title' => 'required|unique:posts|max:255',
```

**Задължително|уникално:в таблица posts|максимум:255 знака**

```
    'body' => 'required',
```

```
    'publish_at' => 'nullable|date',
```

**null|валидна дата**

```
]);
```

[Подробна информация](#)

Validation Error messages /

съобщения след неуспешна валидация

# Validation Error messages

## Подробна информация

При неуспешна валидация Ларавел автоматично пренасочва обратно.

Всички грешки от валидацията автоматично се добавят към сесията до следващата заявка/[flashed to the session](#)/.

Ларавел проверява за грешки в данните от сесията и ако има такива, автоматично ги свързва към съответното view.

Извеждаме грешките с променливата \$errors /инстанция на Illuminate\Support\MessageBag. [За повече информация](#)/

# Validation Error messages

В нашия пример, потребителя ще бъде върнат обратно към метода create в случай, че валидацията е неуспешна и имаме възможност да отпечатаме съобщенията за грешки в create.bade.php

```
<!-- /resources/views/lecture/create.blade.php -->
```

```
<h1>Create Lecture</h1>
```

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

```
<!-- Create Lecture Form -->
```

# Custom Error Messages

# Custom Error Messages

## Промяна на съобщенията за грешки в Request класовете, които създаваме

Презаписваме messages метода.

Този метод трябва да върне масив от атрибут/правило двойки и съобщенията за грешки, които им принадлежат.

```
/**
 * Get the error messages for the defined validation rules.
 *
 * @return array
 */
public function messages()
{
    return [
        'title.required' => 'A title is required',
        'body.required' => 'A message is required',
    ];
}
```

[Работа с custom error messages](#)



## Laravel Flash Messages

съобщения след успешна валидация на формите

# Laravel Session

Сесии в Ларавел  
как да запазваме и  
използваме данните,  
записани в сесиите

## Извеждане на данни

Два са основните подхода при работа с данни от сесиите в Ларавел.

1. global session helper
2. Чрез инстанция на Request.

# Laravel Session

Достъп до данните от сесията чрез  
инстанция на Request класа,  
която посочваме в метода на контролера./type-hint/  
Зависимостите на методите в контролера се  
инжектират автоматично от Laravel [service container](#):

```
namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Http\Controllers\Controller;

class UserController extends Controller
{
    /**
     * Show the profile for the given user.
     *
     * @param Request $request
     * @param int $id
     * @return Response
     */
    public function show(Request $request, $id)
    {
        $value = $request->session()->get('key');

        //
    }
}
```

# Laravel Session

Проверка дали конкретни данни съществуват в сесията

За да определим дали дадена стойност съществува в сесията използваме метода `has()`.

Той връща `true` ако стойността съществува и не е `null`:

```
if ($request->session()->has('users')) {  
    //  
}
```

# Laravel Session

**Flash Данни** - запазване на данни в сесията до следващата заявка.

Методът `flash()` - данните, запазени с този метод в сесията ще бъдат налични само до следващата HTTP заявка, след което ще бъдат изтрети.

```
$request->session()->flash('status', 'Task was  
successful!');  
return Redirect::to('home');
```

```
//отпечатваме съобщението във вюто с  
@if(Session::has('status')).....
```

# Laravel Session

**Flash съобщенията** обикновено се създават на база действията на потребителя

- За потвърждение от потребителя, че наистина иска да извърши съответното действие.
- отпечатване на съобщения за успешно/неуспешно извършено действие от тях