

Lecture 6.4

Laravel – Models Form Submission Basics and Validation

Course Instructor
Engr. Madeha Mushtaq

Models

- Model works with data.
- For example, Fetching data from the database, inserting data and/or updating data.
- In Laravel, there will be a model for each table of the database.

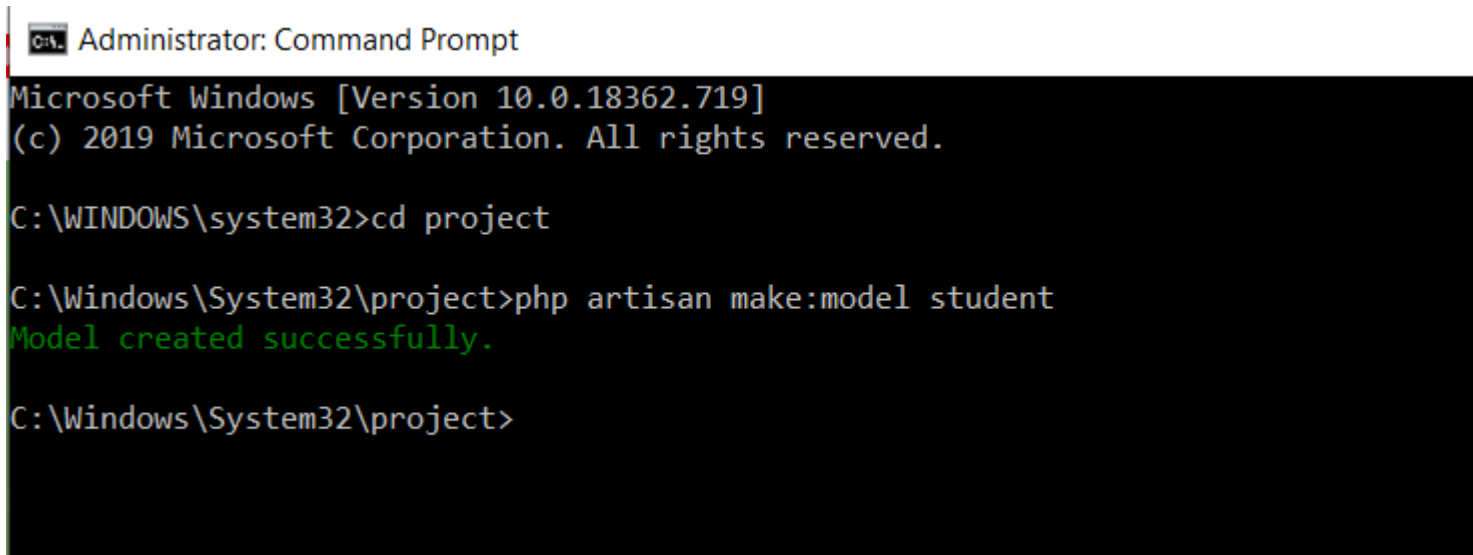
Database Configuration with Laravel

- Open and edit .env file.

```
1 APP_NAME=Laravel
2 APP_ENV=local
3 APP_KEY=base64:YgNIMEXsm4zMdhkIQzJ33ddUfMRnFtmXBA/IHESQ2kM=
4 APP_DEBUG=true
5 APP_URL=http://localhost
6
7 LOG_CHANNEL=stack
8
9 DB_CONNECTION=mysql
10 DB_HOST=127.0.0.1
11 DB_PORT=3308
12 DB_DATABASE=laraveldemo
13 DB_USERNAME=root
14 DB_PASSWORD=
15
```

Model Creation

- For creating model: php artisan make:model name



```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.18362.719]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>cd project

C:\Windows\System32\project>php artisan make:model student
Model created successfully.

C:\Windows\System32\project>
```

Inserting Data in Database

- Inserting data into table.
- Hellocontroller.php

```
1 <?php
2 namespace App\Http\Controllers;
3
4 use Illuminate\Foundation\Auth\Access\AuthorizesRequests;
5 use Illuminate\Foundation\Bus\DispatchesJobs;
6 use Illuminate\Foundation\Validation\ValidatesRequests;
7 use Illuminate\Routing\Controller as BaseController;
8 use App\student;
9
10 class Hellocontroller extends Controller
11 {
12     public function index()
13     {
14         $students = new student;
15         $students -> sname = "Sara";
16         $students -> standard = 12;
17         $students -> save();
18         return view ('hello');
19     }
20 }
21 ?>
```

Or alternatively, can insert data like that:

```
class Hellocontroller extends Controller
{
    public function index()
    {
        $students = new student(['sname' => 'sana', 'standard' => '11']);
        $students -> save();
        return view ('hello');
    }
}
```

Inserting Data in Database

- Our model, student.php

```
1  <?php
2
3  namespace App;
4
5  use Illuminate\Database\Eloquent\Model;
6
7  class student extends Model
8  {
9
10 }
11
```

Inserting Data in Database

- By default, Laravel assume all the tables have timestamp fields - created_at and updated_at.
- If your DB table doesn't have those fields, and you will try to insert data- you will get SQL error. Laravel would try to automatically fill in created_at/updated_at and wouldn't find them.

The screenshot shows a database management interface with a top navigation bar containing icons for Browse, Structure, SQL, Search, Insert, Export, Import, Privileges, Operations, and Triggers. Below the navigation bar, a green message box states: "✓ Table students has been altered successfully." Below this, the SQL command used for the alteration is displayed: `ALTER TABLE `students` CHANGE `updated_at` `updated_at` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP;`

Below the SQL command, a table structure view for the 'students' table is shown. The table has the following columns:

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	int(11)			No	None		AUTO_INCREMENT	Change Drop More
2	sname	varchar(255)	latin1_swedish_ci		No	None			Change Drop More
3	Standard	int(11)			No	None			Change Drop More
4	created_at	timestamp			No	CURRENT_TIMESTAMP			Change Drop More
5	updated_at	timestamp			No	CURRENT_TIMESTAMP			Change Drop More

At the bottom of the interface, there is a toolbar with various actions: Check all, With selected: Browse, Change, Drop, Primary, Unique, Index, Fulltext, and Fulltext. Below the toolbar, there are links for Print, Propose table structure, Move columns, and Normalize.

Inserting Data in Database

- **\$timestamps:**
- By default, \$timestamps = true.
- To disable the automatic timestamps(If we don't want to include created_at and updated_at),
 - set \$timestamps = false.

```
1  <?php
2
3  namespace App;
4
5  use Illuminate\Database\Eloquent\Model;
6
7  class student extends Model
8  {
9      public $timestamps = false;
10 }
11
```


Inserting Data in Database

- **\$fillable:**
- If you do not want to insert value of standard in the database, you can only write sname in fillable.
- A 0 value will be inserted for standard.

```
class student extends Model
{
    protected $fillable = ['sname', 'standard'];
    public $timestamps = false;
}
```

Model Class with different name

- This Teacher model will use “students” table.
- Create a variable in model Teacher.php and assign it the name of the table that you want to use, as shown below:

Teacher Model

```
1 <?php
2
3 namespace App;
4
5 use Illuminate\Database\Eloquent\Model;
6
7 class Teacher extends Model
8 {
9     protected $fillable = ['sname', 'standard'];
10    public $timestamps = false;
11    protected $table = 'students';
12 }
13
```

```
1 <?php
2 namespace App\Http\Controllers;
3
4 use Illuminate\Foundation\Auth\Access\AuthorizesRequests;
5 use Illuminate\Foundation\Bus\DispatchesJobs;
6 use Illuminate\Foundation\Validation\ValidatesRequests;
7 use Illuminate\Routing\Controller as BaseController;
8 use App\student;
9 use App\Teacher;
10
11 class Hellocontroller extends Controller
12 {
13     public function index()
14     {
15         $students = new Teacher;
16         $students->sname = "Ahad";
17         $students->standard = 9;
18         $students->save();
19         return view('hello');
20     }
21 }
22
23 ?>
```

Getting Data from Form

- Request object:
- It contains all the data that came across with the HTTP request, such as headers, POST data, query string arguments, etc.
- You can use that object to access whatever request information you need to make decisions in your controllers.

Form Submission Basics

- Hellocontroller.php

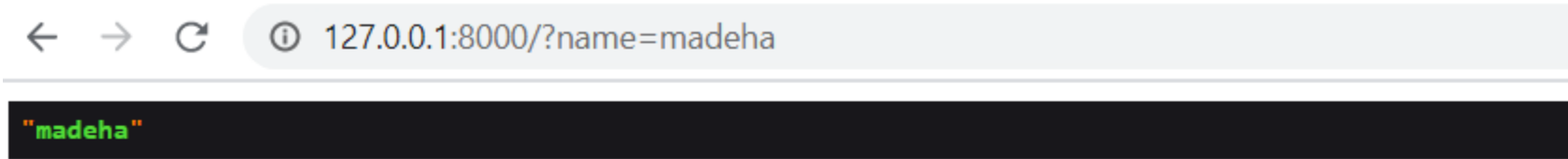
```
1 <?php
2 namespace App\Http\Controllers;
3
4 use Illuminate\Foundation\Auth\Access\AuthorizesRequests;
5 use Illuminate\Foundation\Bus\DispatchesJobs;
6 use Illuminate\Foundation\Validation\ValidatesRequests;
7 use Illuminate\Routing\Controller as BaseController;
8 use App\student;
9 use App\Teacher;
10 use Illuminate\Http\Request;
11
12 class Hellocontroller extends Controller
13 {
14     public function index(Request $request)
15     {
16         dd($request -> name);
17     }
18 }
19 ?>
```

- Dd() is a die and dump function.
- We can also use echo instead

```
class Hellocontroller extends Controller
{
    public function index(Request $request)
    {
        echo $request -> name;
    }
}
?>
```

Form Submission Basics

- Put any name you want to display:



Form Submission Basics

- Request object can be used in a number of ways with dd():

```
class Hellocontroller extends Controller
{
  public function index(Request $request)
  {
    dd($request -> all());
  }
}
```

← → ↻ ⓘ 127.0.0.1:8000/?name=madeha&college=UET&CellNum=123456789

```
array:3 [▼
  "name" => "madeha"
  "college" => "UET"
  "CellNum" => "123456789"
]
```

Form Submission Basics

```
1 <?php
2 namespace App\Http\Controllers;
3
4 use Illuminate\Foundation\Auth\Access\AuthorizesRequests;
5 use Illuminate\Foundation\Bus\DispatchesJobs;
6 use Illuminate\Foundation\Validation\ValidatesRequests;
7 use Illuminate\Routing\Controller as BaseController;
8 use App\student;
9 use App\Teacher;
10 use illuminate\Http\Request;
11
12 class requestcontroller extends Controller
13 {
14     public function index(Request $request)
15     {
16         echo $request -> get('name', 'Name not entered');
17     }
18 }
19 ?>
```

```
class Hellocontroller extends Controller
{
    public function index(Request $request)
    {
        echo $request -> get('name');
    }
}
?>
```

CSRF(Cross Site Request Forgery) Token

- To protect your application, Laravel uses CSRF tokens.
- CSRF tokens are strings that are automatically generated and can be attached to a form when the form is created.
- They are used to uniquely identify forms generated from the server.
- The idea behind it is that when the server receives POST requests, the server checks for a CSRF token.
- If the POST request has a token that matches the active existing CSRF token created by the framework, the form is processed.

CSRF(Cross Site Request Froger) Token

- If not, the form is not processed and an error is sent back to the client making the request.
- This token is also generated per request meaning it cannot be reused once it's used.
- Let's see how this works in practice.

CSRF(Cross Site Request Froger) Token

- Create a view contact.blade.php and a controller contactcontroller.

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Foundation\Auth\Access\AuthorizesRequests;
6 use Illuminate\Foundation\Bus\DispatchesJobs;
7 use Illuminate\Foundation\Validation\ValidatesRequests;
8 use Illuminate\Routing\Controller as BaseController;
9 use App\student;
10 use App\Teacher;
11 use Illuminate\Http\Request;
12 class contactcontroller extends Controller
13 {
14     public function index()
15     {
16         return view('contact');
17     }
18     public function store(Request $request)
19     {
20         dd($request -> all());
21     }
22 }
```

CSRF(Cross Site Request Froger) Token

```
1
2 <form action = "{{route('contactstore')}}" method = "post">
3
4     @csrf
5
6     <label for = "Name"> Name: </label>
7     <input type = "text" name = "name">
8     <input type = "submit">
9 </form>
```

- Update web.php to include the following routes:

```
Route::get('/', 'contactcontroller@index');
Route::post('/contact', 'contactcontroller@store')-> name('contactstore');
```

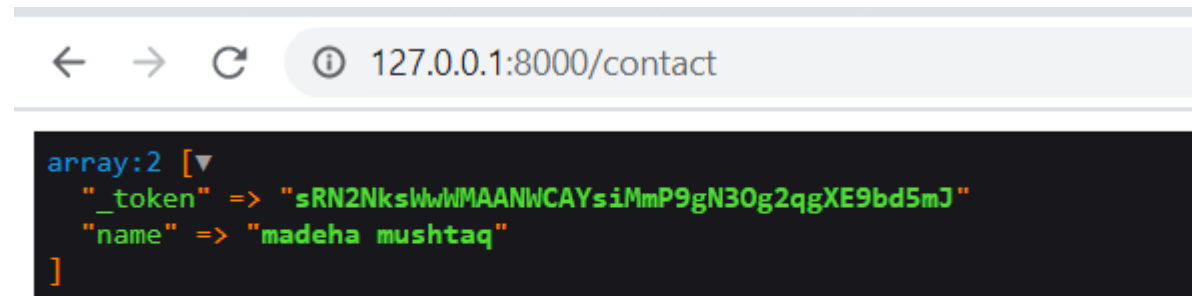
CSRF(Cross Site Request Frogerly) Token

- Output



← → ↻ ⓘ 127.0.0.1:8000

Name:



← → ↻ ⓘ 127.0.0.1:8000/contact

```
array:2 [▼  
  "_token" => "sRN2NkslWwMAANWCAYsiMmP9gN3Og2qgXE9bd5mJ"  
  "name" => "madeha mushtaq"  
]
```

Form Validation in Laravel

- Suppose we want to validate email address.
Contactcontroller.php

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Foundation\Auth\Access\AuthorizesRequests;
6 use Illuminate\Foundation\Bus\DispatchesJobs;
7 use Illuminate\Foundation\Validation\ValidatesRequests;
8 use Illuminate\Routing\Controller as BaseController;
9 use App\student;
10 use App\Teacher;
11 use Illuminate\Http\Request;
12 class contactcontroller extends Controller
13 {
14     public function index()
15     {
16         return view('contact');
17     }
18     public function store(Request $request)
19     {
20         $this->validate($request, ['email'=>'required|email']);
21     }
22 }
```

Form Validation in Laravel

- Contact.blade.php

```
1  @if($errors->any())
2
3      @foreach($errors->all() as $error)
4          <li>{{ $error }}</li>
5      @endforeach
6
7  @endif
8
9  <form action = "{{ route('contactstore') }}" method = "post">
10      @csrf
11      <label for = "Email"> Email: </label>
12      <input type = "text" name = "email">
13      <input type = "submit">
14  </form>
```

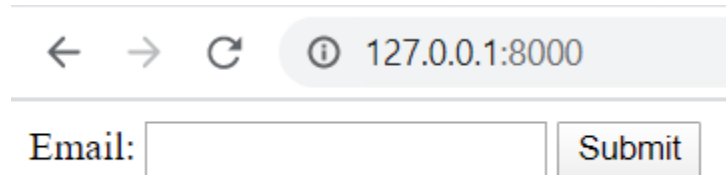
- \$errors is a laravel variable in which errors are stored.

- Web.php

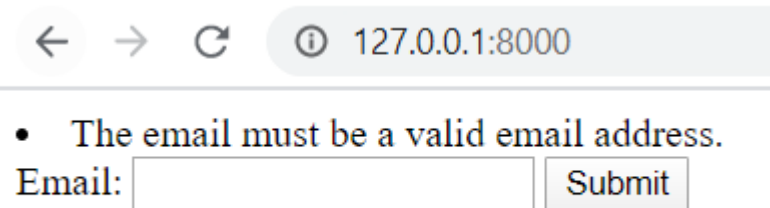
```
Route::get('/', 'contactcontroller@index');

Route::post('/contact', 'contactcontroller@store')-> name('contactstore');
```

Form Validation in Laravel



A screenshot of a web browser window. The address bar shows the URL 127.0.0.1:8000. Below the address bar, there is a form with the label "Email:" followed by an empty text input field and a "Submit" button.



A screenshot of a web browser window showing a validation error. The address bar shows the URL 127.0.0.1:8000. Below the address bar, there is a form with the label "Email:" followed by an empty text input field and a "Submit" button. A red error message is displayed above the input field: "The email must be a valid email address."

If you don't
put valid
email
address.



A screenshot of a web browser window showing a validation error. The address bar shows the URL 127.0.0.1:8000. Below the address bar, there is a form with the label "Email:" followed by an empty text input field and a "Submit" button. A red error message is displayed above the input field: "The email field is required."

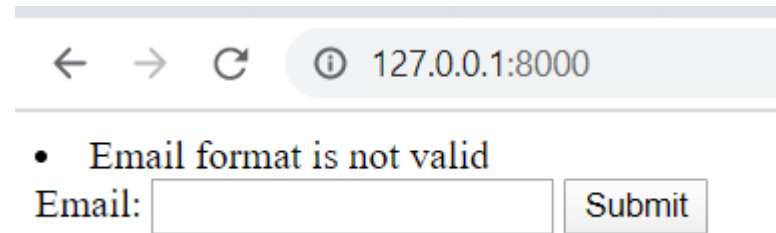
If you leave it
empty.

Form Validation in Laravel

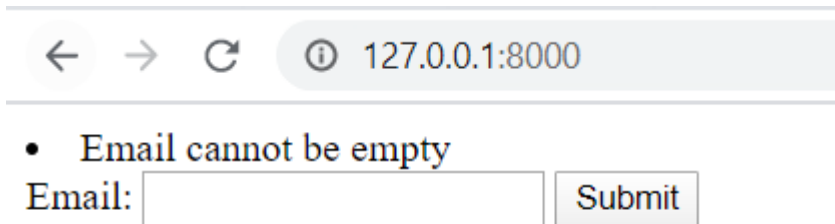
- Creating custom validation messages in Laravel.

```
public function index()
{
    return view('contact');
}
public function store(Request $request)
{
    $this->validate($request, ['email'=>'required|email'],
    ['email.required' => 'Email cannot be empty',
    'email'=> 'Email format is not valid']);
}
```


Form Validation in Laravel



A screenshot of a web browser window. The address bar shows '127.0.0.1:8000'. Below the address bar, there is a list of validation errors: 'Email format is not valid'. Below the error message, there is a form with a label 'Email:' followed by an empty text input field and a 'Submit' button.



A screenshot of a web browser window. The address bar shows '127.0.0.1:8000'. Below the address bar, there is a list of validation errors: 'Email cannot be empty'. Below the error message, there is a form with a label 'Email:' followed by an empty text input field and a 'Submit' button.

- But these messages are local to the contactcontroller. And can not be seen outside.

Form Validation in Laravel

- We can create custom messages globally too, so they are visible to all controllers.
- Go to project\resources\lang\en,
- open validation.php. Edit it.

```
/*
|-----
| Custom Validation Language Lines
|-----
|
| Here you may specify custom validation messages for attributes using the
| convention "attribute.rule" to name the lines. This makes it quick to
| specify a specific custom language line for a given attribute rule.
|
*/

'custom' => [
    'email' => [
        'required' => 'It is required',
        'email' => 'This is not right';
    ],
],
```

Form Validation in Laravel

← → ↻ ⓘ 127.0.0.1:8000

- This is not right

Email:

← → ↻ ⓘ 127.0.0.1:8000

- It is required

Email:

Login System in Laravel

- Create a database in Phpmyadmin.
- Edit .env file to reflect the name of the database.
- On cmd type “php artisan migrate”.
- It will create tables for you in the database.
- Run composer require laravel/ui
- Finally use this command: php artisan ui vue --auth

Login System in Laravel





```
-> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi
Discovered Package: facade/ignition
Discovered Package: fideloper/proxy
Discovered Package: fruitcake/laravel-cors
Discovered Package: laravel/tinker
Discovered Package: laravel/ui
Discovered Package: nesbot/carbon
Discovered Package: nunomaduro/collision
Package manifest generated successfully.
26 packages you are using are looking for funding.
Use the `composer fund` command to find out more!

C:\Windows\System32\project>php artisan ui vue --auth
Vue scaffolding installed successfully.
Please run "npm install && npm run dev" to compile your fresh scaffolding.
Authentication scaffolding generated successfully.




C:\Windows\System32\project>
```

Login System in Laravel

- In views folder, it will create an auth folder.
- It will create for you all the views related to authentication.

This PC > Windows (C:) > Windows > System32 > project > resources > views > auth				
Name	Date modified	Type	Size	
 passwords	12/04/2020 4:11 PM	File folder		
 login.blade.php	12/04/2020 4:11 PM	PHP File	4 KB	
 register.blade.php	12/04/2020 4:11 PM	PHP File	4 KB	
 verify.blade.php	12/04/2020 4:11 PM	PHP File	2 KB	

Login System in Laravel

PC > Windows (C:) > Windows > System32 > project > resources > views > auth > passwords			
Name	Date modified	Type	Size
 confirm.blade.php	12/04/2020 4:11 PM	PHP File	3 KB
 email.blade.php	12/04/2020 4:11 PM	PHP File	2 KB
 reset.blade.php	12/04/2020 4:11 PM	PHP File	3 KB

Login System in Laravel

← → ↻ ⓘ 127.0.0.1:8000

☆ ≡ M ⋮

LOGIN REGISTER

← → ↻ ⓘ 127.0.0.1:8000/login

Laravel

- [Login](#)
- [Register](#)

Login

E-Mail Address

Password

☐ Remember Me

Login [Forgot Your Password?](#)

← → ↻ ⓘ 127.0.0.1:8000/register

Laravel

- [Login](#)
- [Register](#)

Register

Name

E-Mail Address

Password

Confirm Password

Register

Class Task

1. A company uses the following Laravel routes to get employees information:

```
Route::get('/employee/{id?}', function ($id = 0) {  
    return view('employee');  
});  
Route::get('/employee/{id}/subordinates/', function ($id) {  
    return view('subordinates');  
});  
Route::get('/employee/{id}/subordinates/{subordinateId}', function ($id,  
$subordinateId) {  
    return view('subordinateDetails');  
});
```

Class task

A request to `/employee/` will get mapped to the _____ view.

A request to `/employee/12` will get mapped to the _____ view.

A request to `/employee/10/subordinates/1` will get mapped to the _____ view.

A request to `/employee/10/subordinates/` will get mapped to the _____ view.

A request to `/employee/10/subordinates/1/10` will get mapped to the _____ view.