Laravel: Database access & Models – Part 3

DWES

6.- Input Data

We have seen how to insert and retrieve data from the database. Next, we will see how to collect data from a form to be able to insert or modify data.

When doing an HTTP Request, Laravel uses the dependency injector, we can use it by adding the Request class to the constructor or method of the controller in which it is needed. Laravel will automatically inject all the code necessary for use. Here is an example:

```
class UserController extends Controller
{
        public function store(Request $request)
        {
                $name = $request->input('nombre');

                //...
        }
}
```

6.- Input Data

You can see how the Request $request parameter is passed to the store method and from there in that method you can use the $request variable to obtain the input data whether received by form or by query string. To pass more parameters they will have to be added after the dependencies:

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

6.- Input Data

Using the input method, the value of the variable is obtained, and a default value can also be indicated in case the variable is empty. Additionally, it can also be accessed directly with the variable name:

```
$name = $request->input('nombre');
$name = $request->input('nombre', 'Pedro');
$name = $request->nombre;
```

To check if a value exists, the has method is used:

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

6.- Input Data

Using the following methods, an array with the indicated variables is obtained:

```
// Obtener todos:
$input = $request->all();

// Obtener solo los campos indicados:
$input = $request->only('username', 'password');

// Obtener todos excepto los indicados:
$input = $request->except('credit_card');
```

6.- Input Data

When working with forms that contain array inputs, use "dot" notation to access the arrays:

```php
$name = $request->input('products.0.name');

$names = $request->input('products.*.name');
```

6.- Input Data

Retrieving Uploaded Files

You may retrieve uploaded files from a Request using the file method. It returns an instance of the Illuminate\Http\UploadedFile class and provides a variety of methods for interacting with the file:

```php
$file = $request->file('photo');

$file = $request->photo;
```

6.- Input Data

Retrieving Uploaded Files

You may determine if a file is present on the request using the hasFile method:

```
if ($request->hasFile('photo')) {
    // ...
}
```

In addition to checking if the file is present, you may verify that there were no problems uploading the file via the isValid method:

```
if ($request->file('photo')->isValid()) {
    // ...
}
```

6.- Input Data

Retrieving Uploaded Files

To store an uploaded file, you will use one of your configured filesystems. The store method will move an uploaded file to one of your disks, it accepts the path where the file should be stored relative to the filesystem's configured root directory, also accepts an optional second argument for the name of the disk that should be used to store the file:

```php
$path = $request->photo->store('images');

$path = $request->photo->store('images', 's3');
```

6.- Input Data

Retrieving Uploaded Files

If you do not want a filename to be automatically generated, you may use the storeAs method, which accepts the path, filename, and disk name as its arguments:

```
$path = $request->photo->storeAs('images', 'filename.jpg');

$path = $request->photo->storeAs('images', 'filename.jpg', 's3');
```

7.- User control

Laravel provides two optional packages to assist you in managing authenticating requests. The recommended one is Breeze. To install it run:

```
composer require laravel/breeze --dev
```

Now run the following command, it publishes the authentication views, routes, controllers, and other resources to your application:

```
php artisan breeze:install


php artisan migrate
npm install
npm run dev
```

7.- User control

When asked, you can choose the option that best suits your project, in this case I have selected "blade"

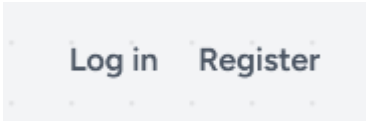
```
  Which Breeze stack would you like to install?
  Blade with Alpine ..................................................... blade
  Livewire (Volt Class API) with Alpine ................................. livewire
  Livewire (Volt Functional API) with Alpine ............................ livewire-functional
  React with Inertia .................................................... react
  Vue with Inertia ...................................................... vue
  API only .............................................................. api
❯ blade
```

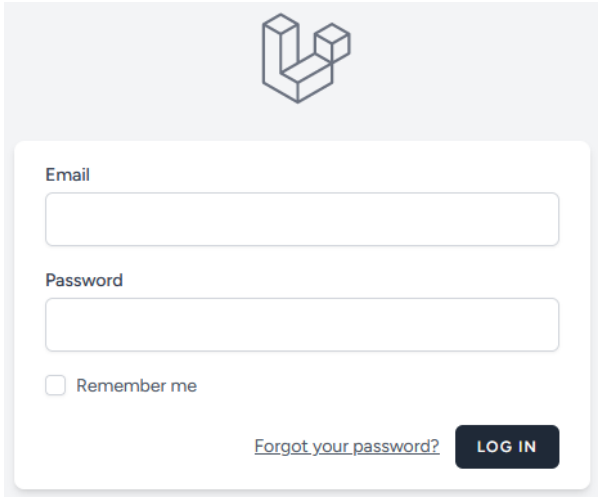
7.- User control

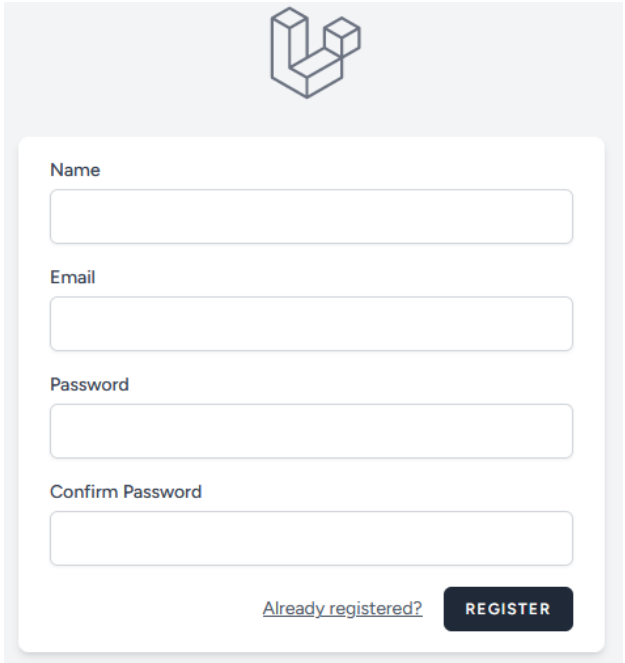
On installation:

- Controllers are created in App\Http\Controllers\Auth\
- Routes are created in routes\auth.php
- Views are created in resources\views\auth\

The application is ready to use the Authentication System:







7.- User control

You may access the authenticated user via the Auth facade's user method:

```php
use Illuminate\Support\Facades\Auth;

// Retrieve the currently authenticated user...
$user = Auth::user();

// Retrieve the currently authenticated user's ID...
$id = Auth::id();
```

7.- User control

Alternatively, once a user is authenticated, you may access the authenticated user via a Request Instance:

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\RedirectResponse;
use Illuminate\Http\Request;

class FlightController extends Controller
{
    /**
     * Update the flight information for an existing flight.
     */
    public function update(Request $request): RedirectResponse
    {
        $user = $request->user();

        // ...

        return redirect('/flights');
    }
}
```

7.- User control

To determine if the user making the incoming HTTP request is authenticated, you may use the check method:

```php
use Illuminate\Support\Facades\Auth;

if (Auth::check()) {
    // The user is logged in...
}
```

A middleware can be used to only allow authenticated users to access a given route:

```php
Route::get('/flights', function () {
    // Only authenticated users may access this route...
})->middleware('auth');
```

7.- User control

Manually Authenticating Users

You are not required to use the authentication scaffolding included with Laravel's application starter kits.

The attempt method is normally used to handle authentication attempts from your application's "login" form.

The attempt method accepts an array of key / value pairs as its first argument. The values in the array will be used to find the user in your database table.

7.- User control

Manually Authenticating Users

So, in the next example, the user will be retrieved by the value of the email column. If the user is found, the hashed password stored in the database will be compared with the password value passed to the method via the array.

You should not hash the incoming request's password value, since the framework will automatically hash the value before comparing it to the hashed password in the database.

An authenticated session will be started for the user if the two hashed passwords match.

7.- User control

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Http\RedirectResponse;
use Illuminate\Support\Facades\Auth;

class LoginController extends Controller
{
    /**
     * Handle an authentication attempt.
     */
```

7.- User control

```
    public function authenticate(Request $request): RedirectResponse
    {
        $credentials = $request->validate([
            'email' => ['required', 'email'],
            'password' => ['required'],
        ]);

        if (Auth::attempt($credentials)) {
            $request->session()->regenerate();

            return redirect()->intended('dashboard');
        }

        return back()->withErrors([
            'email' => 'The provided credentials do not match our records.',
        ])->onlyInput('email');
    }
}
```

7.- User control

Logging out

To manually log users out of your application, you may use the logout method provided by the Auth facade. In addition to calling the logout method, it is recommended that you invalidate the user's session and regenerate their CSRF token. After logging the user out, you would typically redirect the user to the root of your application:

```php
public function logout(Request $request): RedirectResponse
{
    Auth::logout();

    $request->session()->invalidate();

    $request->session()->regenerateToken();

    return redirect('/');
}
```

Much more Laravel...

This is as far as we come for learning basic Laravel. We have left out topics like:

- Exception handling
- Security
- Elloquent ORM collections, serialization, factories...
- API Rest creation and usage
- Testing
- And much more...

Use the documentation and the api to learn on your own!!!

And remember, don’t do like that guy

You copied that function without understanding why it does what it does, and as a result your code IS GARBAGE.

AGAIN.

- Linus Torvalds

Exercise

Next, the video club project will be completed by processing the forms and adding the user authentication system.

- Migration of the users table: check that the table is created in the database, if not, execute the migration. You can check in the migration the structure of the table and what is executed.

- User seed: edit the seed file database/seeders/DatabaseSeeder.php adding the seedUsers method that will have to be called from the run method. Modify the file so that two users are created and then run the Artisan command that processes the seeds and checks in phpMyAdmin that those two users have been entered correctly.

Exercise

- Authentication system

When creating the authentication system scaffolding, the login and logout routes will have been added. Check that there are no more routes (for example if you created them at the beginning of the project) or delete them

Add a group middleware to apply auth to catalog routes. Remember that you can check that the routes and the middleware using: php artisan route:list

Modify the LoginController controller so that upon successful login it redirects to the /catalog path.

Modify the login view so that it uses the layout created at the beginning of the theme. You can create a backup just in case.

Exercise

- Authentication system

Complete the getHome method of the HomeController controller. So that when the user is not logged in it redirects to the /login path.

Modify the navbar view and change the line @if(true || Auth::check()) to @if(Auth::check()) so that the menu is only shown if the user is logged in

Check that the authentication system works. Test registration and login.

Exercise

- Add and edit movies

Edit the routes file and add the necessary routes to collect the data from the forms, they must be in the group protected by the auth middleware:

  - catalog/create: type POST points to the postCreate method of the CatalogController
  - catalog/edit/{id}: type PUT points to the putEdit method of the CatalogController.

Edit the catalog/edit view:

  - The form submission method must be PUT
  - Modify the inputs so that they show the value of the movie {{$pelicula->tittle}}

Exercise

- Add and edit movies

Update the CatalogController with two methods that will use $request dependency injection as an input parameter:

- postCreate method:
  - create an instance of Movie
  - Assign field values to object properties
  - Save the object to the database with the save method
  - Redirect to /catalog path

- putEdit method:
  - Retrieve the movie to an object based on the received identifier
  - Update the values with the data received
  - Save the object to the database with the save method
  - Redirect to the page with the detailed view of the edited movie

Exercise

At this point, the video store application should be completely finished.

Check the operation of the entire application. If you detect any incorrect functionality or something missing, fix it or finish it.