Chapter 21 – Login System (unofficial version)

Laravel authentication systems

Most frameworks provide a library or package to handle authentications. Even if it is an unopinionated framework, you can find third-party libraries for these types of systems.

Laravel 6.* and 7.* provided an official authentication system called *laravel/ui* (https://laravel.com/docs/7.x/authentication). *laravel/ui* is a straightforward authentication system built on the Bootstrap CSS framework (https://github.com/laravel/ui). This library was created by Taylor Otwell (the creator of Laravel). Later, this library provided support to create an authentication system with Vue and React. However, *laravel/ui* is no longer the official authentication system for Laravel 8.* and beyond.

Laravel 11.* have five official authentication systems (https://laravel.com/docs/11.x/starter-kits):

- **Breeze & Blade:** is a simple authentication system based on Blade templates styled with Tailwind CSS.
- **Breeze & Inertia:** Breeze also offers an Inertia.js frontend implementation powered by Vue or React.
- **Breeze & Livewire:** Laravel Breeze also offers Livewire scaffolding. Livewire is a powerful way of building dynamic, reactive, front-end UIs using just PHP.
- **Breeze and Next.js / API:** Laravel Breeze can also scaffold an authentication API that is ready to authenticate modern JavaScript applications such as those powered by Next, Nuxt, and others.
- **Jetstream:** augments functionalities with more robust features and additional frontend technology stacks. Jetstream is designed using Tailwind CSS and offers your choice of Livewire or Inertia.js driven frontend scaffolding.

If we want to use some of the previous options to implement the authentication system, we will need to use additional CSS frameworks or JavaScript frameworks (such as React or Vue) which is out of the scope of this book. Fortunately, *laravel/ui* continues to be available and supports for Laravel 11.* applications.

Installing laravel/ui authentication system

Installing laravel/ui

In the Terminal, go to the project directory, and execute the following:

Execute in Terminal

```
composer require laravel/ui
```

The previous command includes the *laravel/ui* library over our *composer.json* file. And it installs that library in our *vendor* folder.

We also need to generate the frontend scaffolding and the login system. In the Terminal, go to the project directory, and execute the following:

Execute in Terminal

```
php artisan ui bootstrap --auth
```

You will be asked three questions, **type "no"** to the first two questions, and **type "yes"** to the "Controller.php" question (see Fig. 21-1). **There is a warning saying to run "npm" commands, ignore that warning**.

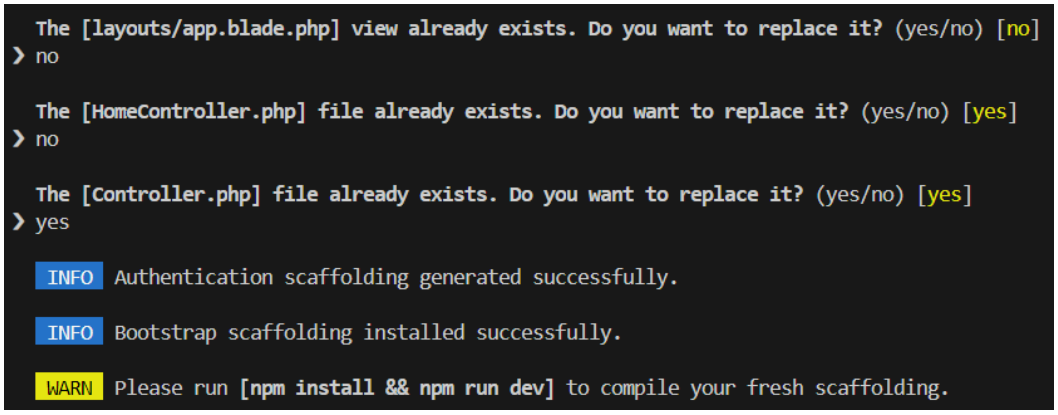

Figure 21-1. Configuring laravel/ui.

laravel/ui creates:

- The *app/Http/Controllers/Auth* folder that includes some authentication controllers such as *LoginController* and *RegisterController*.
- The *resources/views/auth* folder that includes some authentication views such as *login* and *register*. Those views are generated with simple HTML and Bootstrap code.
- A modification over the *web.php* file that includes the *Auth* routes and a */home* route.
- A modification over the *Controller.php* file that includes some authorization and validations methods.

Customizing the authentication system

Let's make some changes to finalize the inclusion of the authentication system over our Online Store application.

Modifying web.php

In *routes/web.php*, remove the following line in **bold** (at the end of the file).

Modify Bold Code

```
...

Auth::routes();
```

```
Route::get('/home', [App/Http/Controllers/HomeController::class, 'index'])->name('home');
```

laravel/ui creates a default (“/home”) route. However, our application does not have a (“/home”) route, so we remove it.

Modifying Auth/LoginController

In *app/Http/Controllers/Auth/LoginController.php*, we replace the *$redirectTo* attribute value, for the next one in **bold**.

Modify Bold Code

```
<?php
…
    use AuthenticatesUsers;

    /**
     * Where to redirect users after login.
     *
     * @var string
     */
    protected $redirectTo = '/';
…
```

The previous attribute is used by Laravel auth to redirect users after login. Since we don’t have a (“/home”) route, we replace it with the main route (“/”).

Modifying Auth/RegisterController

In *app/Http/Controllers/Auth/RegisterController.php*, we replace the *$redirectTo* attribute value, for the next one in **bold**.

Modify Bold Code

```
<?php
…
    use RegistersUsers;

    /**
     * Where to redirect users after registration.
     *
     * @var string
     */
```

```
   protected $redirectTo = '/';
...
```

The previous attribute is used by Laravel auth to redirect users after registration. Since we don't have a ("/home") route, we replace it with the main route ("/").

Modifying app.blade.php

In *resources/views/layouts/app.blade.php*, make the following changes in **bold**.

Modify Bold Code

```
...
        <div class="navbar-nav ms-auto">
          ...
          <a class="nav-link active" href="{{ route('home.about') }}">About</a>
          <div class="vr bg-white mx-2 d-none d-lg-block"></div>
          @guest
          <a class="nav-link active" href="{{ route('login') }}">Login</a>
          <a class="nav-link active" href="{{ route('register') }}">Register</a>
          @else
          <form id="logout" action="{{ route('logout') }}" method="POST">
            <a role="button" class="nav-link active"
               onclick="document.getElementById('logout').submit();">Logout</a>
            @csrf
          </form>
          @endguest
        </div>
...
```

Blade provides *@auth* and *@guest* directives to determine if the current user is authenticated (*@auth*) or is a guest (*@guest*). If the user is a guest, we will show the *register* and *login* links. On the other hand, if the user is authenticated, we will show the *logout* link. All these links are connected to *auth* routes.

Modifying buttons in views/auth/login and views/auth/register

In *resources/views/auth/login.blade.php*, make the following changes in **bold**.

Modify Bold Code

```
...
                        <div class="col-md-8 offset-md-4">
```

```
                        <button type="submit" class="btn bg-primary text-white">
                            {{ __('Login') }}
                        </button>

                        @if (Route::has('password.request'))
                            <a class="btn bg-primary text-white" href="{{ route('password.request') }}">
                                {{ __('Forgot Your Password?') }}
                            </a>
                        @endif
                    </div>
...
```

In *resources/views/auth/register.blade.php*, make the following changes in **bold**.

Modify Bold Code

```
...
                <div class="form-group row mb-0">
                    <div class="col-md-6 offset-md-4">
                        <button type="submit" class="btn bg-primary text-white">
                            {{ __('Register') }}
                        </button>
                    </div>
                </div>
...
```

Running the app

In the Terminal, go to the project directory, and execute the following:

Execute in Terminal

```
php artisan serve
```

Go to the ("/register") route, you will see the registration form and the new links over the navigation bar (see Fig. 21-2). First, create a new user. Then, you will be redirected to the home page as an authenticated user. Then, you will see the *logout* link (see Fig. 21-3). Finally, you can click the *logout* link and test the *login* system with your user credentials (see Fig. 21-4).

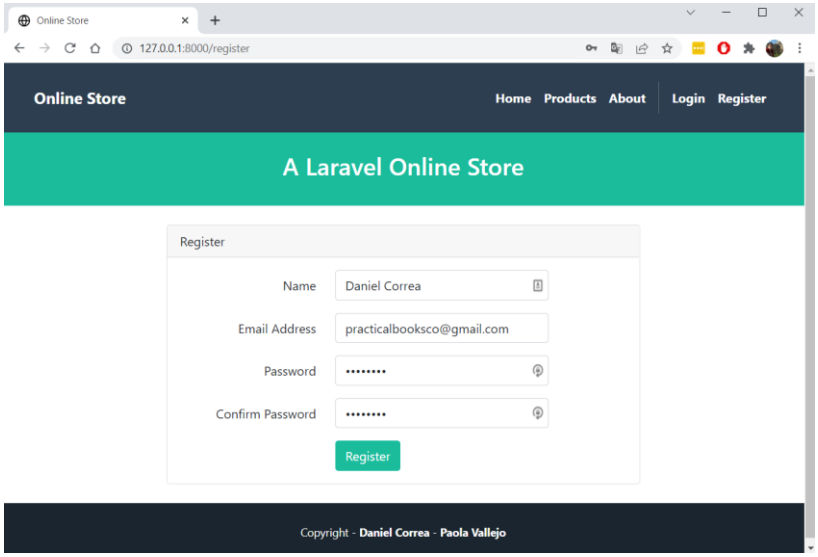

Figure 21-2. Online Store – Register page.

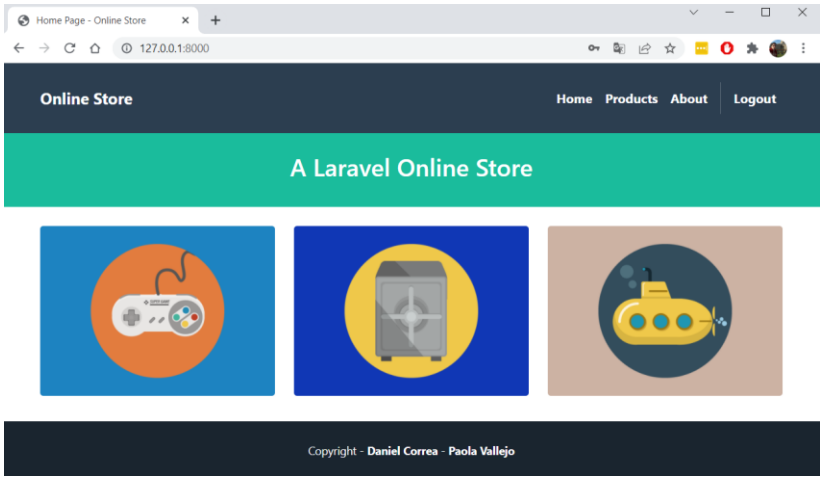

Figure 21-3. Online Store – Home page for authenticated users.

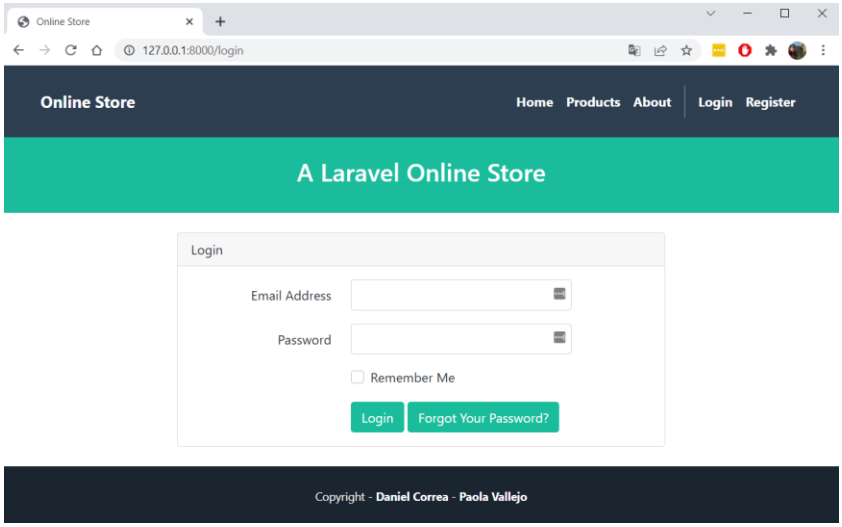

Figure 21-4. Online Store – Login page.

Users are registered in the *users* table. This table was created the first time we ran Laravel Migrations. Laravel also includes a *User* model.