UNIT 6: MODELS II AND FORMS

On the blog project of the previous unit, we are going to add these changes:

Create a one-to-many relationship between the 'User' model and the 'Post' model , both already existing in the application, so that a post is from one user, and a user can have many posts. You must define a new modification migration on the posts table that adds a new field 'user_id', and establish from it the relationship, as we have done in the library example with authors and books.

Create from phpMyAdmin a series of test users in the users table, and associate some of them with the posts that exist.

Modify the 'posts/index.blade.php' view so that, next to the title of each post, in parentheses, the login of the user who created it appears

Now we will add the following:

Create a seeder called 'SeederUsers', with an associated factory called ' UserFactory' (rename the default 'UserFactory' to take advantage of it). Create with this 3 test users, with logins that are unique and one word (use the faker), and passwords also of a single word, unencrypted (to be able to identify them later, if necessary).

Create another seeder called 'PostsSeeder' with an associated factory called 'PostFactory'. In the factory, define with the faker random titles (sentences) and random contents (long texts). Use the seeder to create 3 posts for each of the existing users.

Use the 'php artisan migrate:fresh --seed' option to delete all previous content and fill the database with these new elements. Check then from the page of the list of posts, and from phpMyAdmin, that the information that appears is correct

Create a form to register new posts, in the 'resources/views/posts/create.blade.php' view. Add a couple of fields (a short text and a long text) to fill in the title and content, and a select list to choose the user you are. Remember to define the 'store' method in the post Controller to register the post, and then redirect to the main list of posts. To load the form, add a new option in the main navigation menu (partials.nav).

Validate the post data. In particular, these requirements must be met: The title of the post must be mandatory, and at least 5 characters long The content of the post must be mandatory, and at least 50 characters long. Define custom error messages for each potential validation failure, and display them next to each affected field, as in the library example. In addition, it uses the 'old' function to remember the correct old value, in case one field passes validation but another does not.

In the tab of a post, add a button with a form to delete the post. You must define the code of the 'destroy' method to delete the post and redirect back to the list.

Now we will add the edit form of a post, also from the post tab view. The form must show the data already filled in from the post. This form is loaded from the 'edit' method (which must render the view with the edit form, 'resources/views/posts/edit.blade.php'), and the form will be sent to the 'update' method of the controller, passing as a parameter the id of the post to be modified.