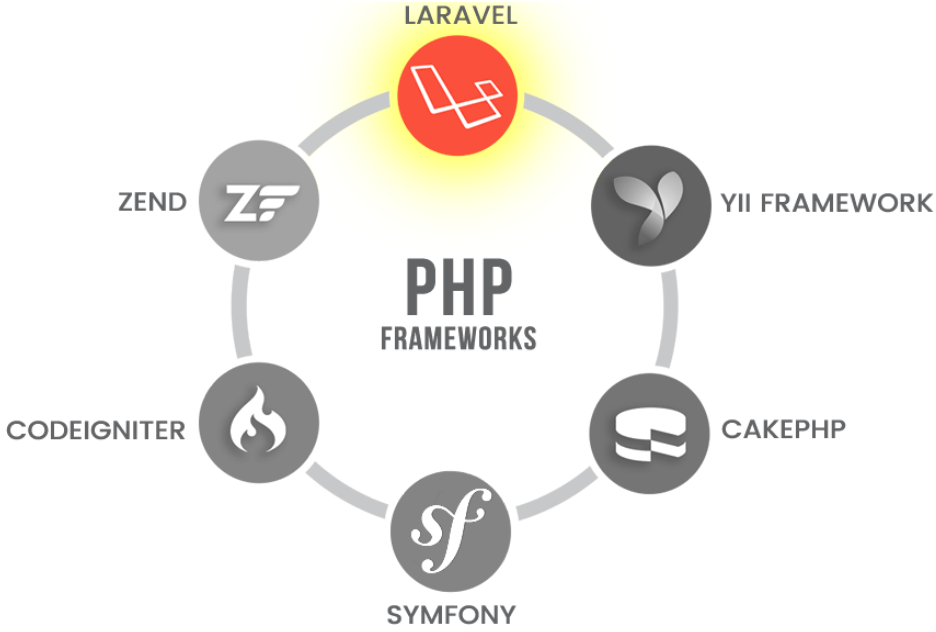
unidad didáctica 7

Laravel - modelo de datos (actividades)



1. ejercicios propuestos

1.1. parte I

1.1.1. Ejercicio 1

Sobre el proyecto **blog** de la sesión anterior, vamos a añadir estos cambios:

- Crea una base de datos llamada `blog` en tu servidor de bases de datos a través de *phpMyAdmin*. Modifica también el archivo `.env` del proyecto para acceder a dicha base de datos con las credenciales adecuadas, similares a las del ejemplo de la biblioteca (cambiando el nombre de la base de datos).
- Elimina todas las migraciones existentes, salvo la de *create_users_table*. Edita esta migración de la tabla usuarios para dejarla igual que el ejemplo de la biblioteca (únicamente con los campos *login* y *password*, además del *id* y los *timestamps*).
- Crea una nueva migración llamada `crear_tabla_posts`, que creará una tabla llamada `posts` con estos campos:
    - Id autonumérico
    - Título del post (`string`)
    - Contenido del post (`text`)
    - *Timestamps* para gestionar automáticamente la fecha de creación o modificación del post
- Lanza las migraciones y comprueba que se crean las tablas correspondientes con los campos asociados en la base de datos.
- Opcionalmente, puedes desactivar las migraciones de Sanctum en el proyecto Laravel, como se ha explicado al final de este documento, si al crear las migraciones aparece alguna otra tabla no deseada, como la de los *personal_access_tokens*. Recuerda ejecutar `php artisan migrate:fresh` de nuevo para borrar el rastro de esas tablas.

1.2. Ejercicio 2

Continuamos con el proyecto **blog** anterior. Modifica si no lo has hecho aún el modelo `User` que viene por defecto para que se llame `Usuario`, igual que hemos hecho en el ejemplo de la biblioteca. Crea un nuevo modelo llamado `Post` para los posts de nuestro blog. Asegúrate de que ambos modelos se ubican en la carpeta `App\Models` del proyecto.

Después, modifica los métodos del controlador `PostController` creado en sesiones anteriores, de este modo:

- El método `index` debe obtener todos los posts de la tabla, y mostrar la vista `posts.index` con ese listado de posts.
    - La vista `posts.index`, por su parte, recibirá el listado de posts y mostrará los títulos de cada uno, y un botón `Ver` para mostrar su ficha (`posts.show`).
    - Debes mostrar el listado de posts ordenado por *título* en orden ascendente, y paginado de 5 en 5.
- El método `show` debe obtener el post cuyo `id` se pasará como parámetro, y mostrarlo en la vista `posts.show` .

    - La vista `posts.show` recibirá el objeto con el post a mostrar, y mostraremos el título, contenido y fecha de creación del post, con el formato que quieras.
- El método `destroy` eliminará el post cuyo *id* recibirá como parámetro, y devolverá la vista `posts.index` con el listado actualizado. Para probar este método, recuerda que debes definir un formulario en una vista (lo puedes hacer para cada post mostrado en la vista `posts.index`) que envíe a la ruta `posts.destroy` usando un método *DELETE*, como hemos explicado en un ejemplo anterior.
- Los métodos `create`, `edit`, `store` y `update` de momento los vamos a dejar sin hacer, hasta que veamos cómo gestionar formularios.
- Para simular la inserción y la modificación, vamos a crear dos métodos adicionales en el controlador, que usaremos de forma temporal:
    - Un método llamado `nuevoPrueba`, que cada vez que lo llamemos creará un post con un título al azar (por ejemplo, "Título X", siendo X un entero aleatorio), y un contenido al azar ("Contenido X"). Puedes emplear la función rand de PHP para generar estos números aleatorios para título y contenido.
    - Un método llamado `editarPrueba`, que recibirá como parámetro un *id* y modificará el título y contenido del post otros generados aleatoriamente, como en el punto anterior.
    - Estos dos métodos (especialmente el primero) nos servirán para crear una serie de posts de prueba que luego nos servirán para probar el listado y la ficha de los posts.
- En el archivo `routes/web.php`, recuerda añadir dos nuevas rutas temporales de tipo `get` para probar estas inserciones y modificaciones. La primera puede apuntar a `/posts/nuevoPrueba`, por ejemplo, y la segunda a `/posts/editarPrueba/{id}`. Recuerda también eliminar o editar la restricción `only` de las rutas del controlador que estableciste la sesión anterior, para que no sólo permita las rutas *index*, *show*, *create* y *edit*, y además permita la de *destroy* (o todas las posibles, si quieres, ya que tarde o temprano las utilizaremos).

> **IMPORTANTE**: los métodos `nuevoPrueba` y `editarPrueba` que has creado en `PostController` NO son métodos estándar de un controlador de recursos, y de ninguna manera estarán disponibles a través de `Route::resource` en `routes/web.php`. Por eso debes definir a mano una ruta para cada uno de ellos en ese archivo, a través de `Route::get`, y esas rutas deben definirse ANTES de la de recursos (`Route::resource`) porque de lo contrario no se emparejarán correctamente.

¿Qué entregar?

Como entrega de esta sesión deberás comprimir el proyecto **blog** con todos los cambios incorporados, y eliminando las carpetas `vendor` y `node_modules` como se explicó en las sesiones anteriores. Renombra el archivo comprimido a `blog_04.zip`.

1.3. parte II

1.3.1. ejercicio 3

Sobre el proyecto **blog** de la sesión anterior, vamos a añadir estos cambios:

- Crea una relación *uno a muchos* entre el modelo de `Usuario` y el modelo de `Post`, ambos ya existentes en la aplicación, de forma que un post es de un usuario, y un usuario puede tener muchos posts. Deberás definir una nueva migración de modificación sobre la tabla *posts* que añada un nuevo campo `usuario_id`, y establecer a partir de él la relación, como hemos hecho en el ejemplo con autores y libros.
- Crea desde *phpMyAdmin* una serie de usuarios de prueba en la tabla *usuarios*, y asocia algunos de ellos a los posts que haya.
- Modifica la vista `posts/index.blade.php` para que, junto al título de cada post, entre paréntesis, aparezca el login del usuario que lo creó.

1.3.2. ejercicio 4

Continuamos con el proyecto **blog** anterior. Ahora añadiremos lo siguiente:

- Crea un *seeder* llamado `UsuariosSeeder`, con un factory asociado llamado `UsuarioFactory` (renombra el que viene por defecto `UserFactory` para aprovecharlo). Crea con esto 3 usuarios de prueba, con *logins* que sean únicos y de una sola palabra (usa el *faker*), y passwords también de una sola palabra, sin encriptar (para poderlos identificar después, llegado el caso).
- Crea otro *seeder* llamado `PostsSeeder` con un factory asociado llamado `PostFactory`. En el *factory*, define con el *faker* títulos aleatorios (frases) y contenidos aleatorios (textos largos). Usa el *seeder* para crear 3 posts para cada uno de los usuarios existentes.

Utiliza la opción `php artisan migrate:fresh --seed` para borrar todo contenido previo y poblar la base de datos con estos nuevos elementos. Comprueba después desde la página del listado de posts, y desde *phpMyAdmin*, que la información que aparece es correcta.

1.3.3. ejercicio 5

Opcional

Añade al proyecto **blog** un nuevo modelo llamado `Comentario`, junto con su migración y controlador asociados. Cada comentario tendrá como campo el contenido del comentario, y estará relacionado *uno a muchos* con el modelo `Usuario`, de forma que un usuario puede tener muchos comentarios, y cada comentario pertenece a un usuario. También tendrá una relación *uno a muchos* con el modelo `Post`, de modo que un comentario pertenece a un post, y un post puede tener muchos comentarios. Por lo tanto, la migración de los comentarios deberá tener como campos adicionales la relación con el usuario (`usuario_id`) y con el post al que pertenece (`post_id`).

Aplica la migración para reflejar la nueva tabla en la base de datos, y utiliza un *seeder* y un *factory* para crear 3 comentarios en cada post, con el usuario que sea. A la hora de aplicar todo esto, borra los contenidos previos de la base de datos (`migrate:fresh --seed`).

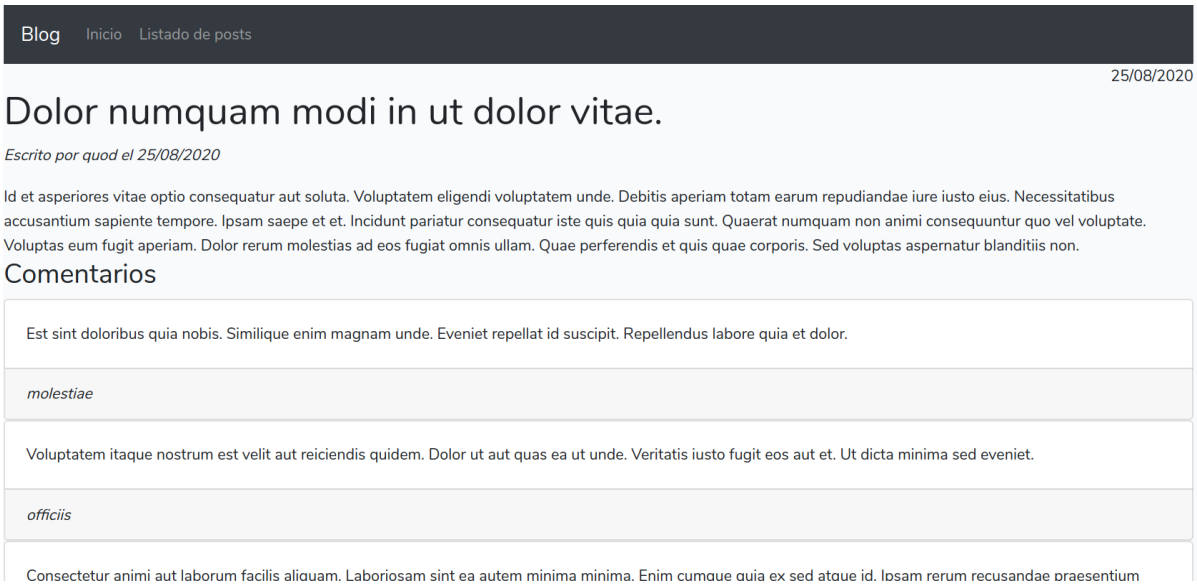
> **AYUDA**: si quieres elegir un usuario al azar como autor de cada comentario, puedes hacer algo así:

```
Usuario::inRandomOrder()->first();
```

> En este caso, sería conveniente que ese usuario aleatorio se añada directamente en el *factory* del comentario, y no en el *seeder*, ya que de lo contrario es posible que genere el mismo usuario para todos los comentarios de un post.

En la ficha de los posts (vista `posts/show.blade.php`), añade el código necesario para mostrar el *login* del usuario que ha hecho el *post*, y el listado de comentarios asociado al post, mostrando para cada uno el *login* del usuario que lo hizo, y el texto del comentario en sí. Utiliza también la librería *Carbon* para mostrar la fecha de creación del post (o la de los comentarios, como prefieras) en formato *d/m/Y*.

Aquí tienes una captura de pantalla de cómo podría quedar:



¿Qué entregar?

Como entrega de esta sesión deberás comprimir el proyecto **blog** con todos los cambios incorporados, y eliminando las carpetas `vendor` y `node_modules` como se explicó en las sesiones anteriores. Renombra el archivo comprimido a `blog_07d.zip`.

2. bibliografia

- Nacho Iborra Baeza.