PROYECTO BLOG

Crea el proyecto BLOG con Laravel y completa los siguientes ejercicios que deberás entregar. Para ello usaremos el contenedor Docker Mysql de la UD9 y el contenedor Docker para Web disponible en Aules, que deberás ponerlo en marcha. Deberás entrar dentro del contenedor con el comando

```
docker compose exec web bash
```

INSTALACIÓN

Instala el instalador de laravel usando composer

```
composer global require laravel/installer
(dejar que instale completamente)
echo "export PATH=$PATH:$HOME/.composer/vendor/bin" >> ~/.bashrc
source ~/.bashrc
laravel --version
```

Ahora ya puedes generar el proyecto blog en la carpeta code del contenedor Docker

```
laravel new blog
chown -R XXXX:XXXX blog (sustituye XXXX por el $UID de tu máquina, DAW es 1004) Nota: paso sólo necesario en Ubuntu
```

Ya podemos usar los comandos de Artisan en esa carpeta del proyecto. Con list podemos ver los comandos disponibles y con --version la versión insatalada de Laravel

```
php artisan list
php artisan --version
```

En el VirtualHost disponible en vhost.conf del contenedor, asegúrate de que DocumentRoot y Directory apuntan a /code/blog/public

En la carpeta del proyecto blog, debes dar permisos a las carpetas bootstrap/cache y storage

```
chmod -R 777 bootstrap/cache
chmod -R 777 storage
```

Por otra parte, debes tener creada la base de datos BLOG en el servidor de MySQL (dwes-dbms)

En el fichero .env del proyecto asegúrate de tener los siguientes datos

```
APP_NAME=Blog
...
DB_CONNECTION=mysql
DB_HOST=dwes-dbms
DB_PORT=3306
DB_DATABASE=blog
DB_USERNAME=root
DB_PASSWORD=
...
SESSION_DRIVER=file
```

Al acceder a la web del proyecto, te debe mostrar la página de bienvenida de Laravel cuando ejecutas **php artisan serve** para poner el servidor en marcha, eso indica que todo ha ido bien.

EJERCICIOS

NOTA: Hay una entrega para cada ejercicio, sólo deberéis subir las carpetas que contengan los ficheros creados o modificado es ese mismo ejercicio. Podéis comprimir el zip y eliminar lo que sobra en el archivo comprimido para mantener la estructura de carpetas del proyecto.

1. **Ejercicio 1 Rutas**: Añade las siguientes rutas a la URL posts. Al acceder a esta ruta (127.0.0.1:8085/posts), deberemos ver un mensaje con el texto "Listado de posts". Añade una ruta parametrizada a posts/{id}, de manera que el parámetro id sea numérico (es decir, sólo contenga dígitos del 0 al 9) y obligatorio. Haz que la ruta devuelva el mensaje "Ficha del post XXXX", siendo XXXX el id que haya recibido como parámetro. Ambas rutas deben tener el nombre post_listado y post_ficha respectivamente.
2. **Ejercicio 2 Plantillas**: Define una plantilla llamada **plantilla.blade.php** en la carpeta de vistas del proyecto (resources/views). Define una cabecera con una sección yield para el título, y otra para el contenido de la página, como la del ejemplo de plantillas Blade. Define en un archivo llamado nav.blade.php en la subcarpeta partials, una barra de navegación que nos permita acceder a estas direcciones de momento:
   - Página de inicio
   - Listado de posts

   Incluye la barra de navegación en la plantilla base que has definido antes. A partir de la plantilla base, define otras dos vistas en una subcarpeta posts , llamadas **posts/listado.blade.php** y **posts/ficha.blade.php**. Como título de cada página pon un breve texto de lo que son (por ejemplo, "Listado posts" y "Ficha post"), y como contenido de momento deja un encabezado h1 que indique la página en la que estamos: "Listado de posts" o "Ficha del post XX", donde XX será el identificador del post que habremos pasado por la URL (y que deberás pasar a la vista). Crea una vista llamada **inicio.blade.php** que muestre el título “Inicio del proyecto BLOG” y contenido un h1 con “Bienvenido/a a la biblioteca de TU_NOMBRE”. Haz que las rutas correspondientes de routes/web.php que ya has definido rendericen (muestren) estas vistas en lugar de devolver texto plano.
3. **Ejercicio 3 Controller**: Crea un controlador de recursos (opción -r) llamado **PostController**, que nos servirá para gestionar toda la lógica de los posts del blog. Asigna la ruta con el método resource para acceder a cada método del controlador, en el archivo routes/web.php . Limita con only las acciones sólo a las funciones de listado (index), ficha (show), creación (create) y edición (edit). Renombra las vistas de listado y ficha de un post a **index.blade.php** y **show.blade.php**, dentro de su carpeta posts, y haz que los métodos correspondientes del controlador de posts rendericen estas vistas. Para los métodos create y edit redirijan a la página de inicio, usando la instrucción redirect. Haz los cambios adicionales que sean convenientes (por ejemplo, en el menú de navegación) para que los enlaces sigan funcionando, y prueba que las cuatro rutas (listado, ficha, creación y edición) funcionan adecuadamente
4. **Ejercicio 4 Migraciones**: Crea una base de datos llamada blog en el SGBD MySQL. Revisa el archivo .env del proyecto para acceder a dicha base de datos con las credenciales adecuadas. Elimina las migraciones relativas a password_resets y failed_jobs, y edita la migración de la tabla usuarios (users) 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

Edita la migración create_users_table para que aparezca usuarios en castellano y renombra la migración a crear_tabla_usuarios (no elimines la fecha de creación) así como la clase interna. Edita el método up para dejar los siguientes campos:

- Id autonumérico
- Login (string)
- Password (string)
- Timestamps

Lanza las migraciones y comprueba que se crean las tablas correspondientes con los campos asociados en la base de datos.

5. **Ejercicio 5 Modelos**: Crea el modelo para Post (indica protected $table=’pots’ para que se asocie a la tabla)y asegúrate de que está en la carpeta app/models junto con el de Usuario (tienes que castellanizar el de Users). Modifica los métodos de PostController:
   - **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 o enlace Ver para mostrar su ficha (posts.show). Debes mostrar el listado de posts ordenado por título en orden ascendente.
   - **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.
   - **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 enlace (lo puedes hacer para cada post mostrado en la vista posts.index) que envíe a la ruta posts.destroy usando un método DELETE (con un formuario), como por ejemplo:

```
<form action="{{ route('libros.destroy', $libro) }}" method="POST">
@method('DELETE')
@csrf
<button>Borrar</button>
</form>
```

6. **Ejercicio 6 Relacionando Modelos**: 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. La migración puede llamarse nuevo_campo_usuario_posts sobre la tabla posts (añade --table=posts)

   Crea en la BD 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.

7. **Ejercicio 7 Seeders**: 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. Por ejemplo este código generaría 2 libros para cada autor:

```
$autores->each(function($autor) {
        Libro::factory()->count(2)->create([
            'autor_id' => $autor->id ])
    });
```

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 el SGBD, que la información que aparece es correcta.

8. **Ejercicio 8 Formularios**: Vamos a crear formularios para la inserción, edición y borrado de datos. Para ello crearemos dos vistas **create.blade.php** y **edit.blade.php**.

   create.blade.php: Debe tener un par de campos (un texto corto y un texto largo) para rellenar el título y el contenido, y como autor o usuario del post de momento deja uno predefinido; por ejemplo, el usuario con id = 1, o el primer autor que encuentres en la base de datos (Usuario::get()->first()). Recuerda definir el método store en el controlador de posts para dar de alta el post, y redirigir después al listado principal de posts. Para cargar el formulario, añade una nueva opción en el menú principal de navegación.

   En la ficha de un post, añade un botón con un formulario para borrar el post. Deberás definir el código del método destroy para eliminar el post y redirigir de nuevo al listado.

   edit.blade.php: Añadiremos un enlace al formulario de edición de un post, también desde la vista de la ficha del post. El formulario deberá mostrar los datos ya rellenos del post. Dicho formulario se carga a partir del método edit (que deberá renderizar la vista con el formulario de edición), y el formulario se enviará al método update del controlador, pasándole como parámetro el id del post a modificar.

9. **Ejercicio 9 Formulario Form Request (validación)**:Crea un Form Request (php artisan make:request nomForm) llamado PostRequest, al que indicaremos que valide los datos del post mediante el vector de opciones introducidos en la función rules. Para ello usa el formato 'required|numeric|min:0' y añadiendo una función messages() que devuelva un vector de mensajes de error con el formato 'titulo.required' => 'El título es obligatorio'. En concreto, deben cumplirse estos requisitos:
   - El título del post debe ser obligatorio, y de al menos 5 caracteres de longitud
   - El contenido del post debe ser obligatorio, y de al menos 50 caracteres de longitud

   Cambia los métodos de PostController para que use PostRequest en lugar de Request. Define mensajes de error personalizados para cada posible error de validación usando la variable $errors de Laravel y muéstralos junto a cada campo afectado. Además, utiliza la función **old('campo')** para recordar el valor antiguo correcto en el caso de que un campo pase la validación pero otro/s no. Para mostrar errores puedes seguir el ejemplo:

```
@if ($errors->has('titulo'))
    <div><font color="red">
        {{ $errors->first('titulo') }}
    </font></div>
@endif
```

10. **Ejercicio 10 Servicio REST con API**: Crea un controlador de tipo api llamado PostController en la carpeta App/Http/Controllers/Api, asociado al modelo Post que ya has creado anteriormente. Rellena los métodos index, show, store, update y destroy para que, respectivamente, hagan lo siguiente:
    - **index** deberá devolver en formato JSON el listado de todos los posts con un código 200
    - **show** deberá devolver la información del post que recibe con un código 200
    - **store** deberá insertar un nuevo post con los datos recibidos con un código 201 y utilizando el validador de posts del ejercicio 9 Form Request. Para el usuario creador del post, pásale como parámetro JSON un usuario cualquiera de la base de datos.
    - **put** deberá modificar los campos del post recibidos con un código 200 y empleando también el validador de posts del del ejercicio 9 Form Request.
    - **destroy** deberá eliminar el post recibido, devolviendo null con un código 204

    Crea una colección en Postman llamada Blog que defina una petición para cada uno de los cinco servicios implementados. Comprueba que funcionan correctamente y exporta la colección a un archivo. Debes entregar también el archivo con la colección exportada

NOTA: para optar a una calificación de 5, se deben realizar los ejercicios del 1 al 7 obligatoriamente y el ejercicio 10 se puntúa con 2,5 puntos.

Los ejercicios 8 y 9 son para subir nota y optar a la calificación de 10.

Puntuación individual ejercicios:

Ejercicio1: 0,2 puntos

Ejercicio 2: 1,2 puntos

Ejercicio 3: 0,9 puntos

Ejercicio 4: 0,3 puntos

Ejercicio 5: 0,8 puntos

Ejercicio 6: 0,6 puntos

Ejercicio 7: 1 punto

Ejercicio 8: 1,2 puntos

Ejercicio 9: 1,3 puntos

Ejercicio 10: 2,5 puntos