**LARAVEL 9.0**

1. Para la creación de una aplicación esta se realiza mediante el comando:

composer create-project "laravel/laravel:^9.0" aprendiendo-laravel --prefer-dist

1. Para verificar que la aplicación se ha creado y Laravel esta corriendo se visita el sitio del proyecto en este caso es:

<http://localhost/Master-php/aprendiendo-laravel/public/>

1. El controlador genérico de la aplicación esta en la ruta:

aprendiendo-laravel/app/Http/Controllers/Controller.php

1. Y la vista que se presenta al inicio del proyecto esta en la carpeta:

aprendiendo-laravel/resources/views/welcome.blade.php

1. Y la vista anterior esta asociada a un controlador y en este caso esta asociada a una ruta que está definida en:

aprendiendo-laravel/routes/web.php

1. En esta definición de las rutas me esta allí retornando una vista y lo que hace es acceder a la carpeta views y allí retorna la vista welcome

Route::get('/', function () {

    return view('welcome');

});

1. Creación de Host Virtuales:

Archivos que se configuran:

C:\Windows\System32\drivers\etc\

En el archivo host se adiciona lo siguiente: 127.0.0.1 aprendiendo-laravel.com.devel

C:\wamp64\bin\apache\apache2.4.62.1\conf\extra

En el archivo httpd-vhosts.conf se adiciona lo siguiente:

<VirtualHost \*:80>

DocumentRoot "${INSTALL\_DIR}/www/Master-php/aprendiendo-laravel/public"

ServerName aprendiendo-laravel.com.devel

</VirtualHost>

1. Estructura de un proyecto de Laravel.
2. App: Directorio donde esta el contenido principal de la aplicación tanto de laravel como el HTML de la pagina que vayamos creando.
3. Bootstrap: Directorio que contine un archivo que inicia el framework y también tiene una cahe.
4. Config: Directorio de configuración de la aplicación.
5. Databse: Incluye las migraciones (Permiten versionar los cambios que se hacen a la base de datos) y los seeders que sirven para poblar la base de datos.
6. Public: Directorio que contiene archivos públicos.
7. Resources: Permite tener las vistas de la aplicación.
8. Routes: Tenemos acá todas las rutas de nuestra aplicación.
9. Storage: Almacenamos archivos para cuando subamos archivos.
10. Test: Directorio para hacer test unitarios.
11. Vendor: Es donde se guardan todos los paquetes y librerías que se crean con los comandos de composer.
12. .env: Permite realizar configuraciones a la base de datos.
13. .gitignore: Permite definir que archivos voy a ignorar o no para cuando se suba el proyecto a Git.
14. Artisan: Es usado para ejecutar comandos.
15. Composer.json: Acá esta la configuración del proyecto de Laravel a nivel de dependencias.
16. Package.json: Es a nivel de javascript donde se definen que dependencias se usan.
17. Phpunit.xml: Configuraciones para las pruebas unitarias.
18. Comandos de artisan: Para ejecutar estos comando se puede a través del CMD o iendo a la carpeta y haciendo uso de Open Git Bash here.

*php artisan help*: Esto me permite ver todos los comando básicos que tiene que Laravel.

*php artisan list*: Esto mepermite ver todos los comandos mas importantes, por ejemplo para generar código de diferentes tipos, comandos para borrar cache, comandos para rutas, etc.

*php artisan route:list* Este commando me permite ver todas las ruta que tengo en mi proyecto.

*php artisan make:controller PruebasController*  Esto em permite crear un controller llamado PruebasController.

1. Laravel utiliza Blade para trabajar con las vistas y con las plantillas. Nos permite herencia de plantillas, uso de plantillas base. En nuestro archivos podemos hacer uso de interpolación de variables para imprimir su contenido.

Esto es válido si uso los caracteres de inicio de php y al final:

<h1><?=$titulo?></h1>

Con Blade se nos permite también imprimir así con el uso de doble llave, para que el usuario logre identificar si está modificando una vista o un controlador:

<h1>{{$titulo}}</h1>

1. En el uso de Layout se usa @yield('titulo') para definir una sección que puede ser rellenada desde otra vista hija. Otro bloque que se usa es @section('header') y la diferencia con yield es que con yield no tiene un contenido predeterminado y section si puede tener un contenido predeterminado, y en ambos bloques se pueden sustituir o agregar contenido.
2. Para crear un Controller lo realizamos a trave del siguiente comando dentro del proyecto:

php artisan make:controller PeliculaController

1. Para crear un controlador resource (Son controladores completos para operaciones CRUD – Crear, Leer, Actualizar y Eliminar ) se realiza mediante el siguiente comando

php artisan make:controller UsuarioController --resource

1. Middlewares o Filtros. Es un componente que nos permite filtrar las peticiones que hacemos mediante http, de forma que es un filtro que nos permite controlar o hacer una lógica antes de visitar una página o un controlador. Estos se crean mediante el comando:

php artisan make:middleware TestYear

Los middleware se guardan en la carpeta: app/Http/Middleware/

Para hacer el uso del middleware este se define en la ruta (web.php)

También se deben de dar de alta y ello se hace en el archivo kernel.php que se encuentra en la ruta app/Http y se definen allí en la sección de $routeMiddleware así:

‘testyear’ => \App\Http\Middleware\TestYear::class;

**BASES DE DATOS Y QUERY BUILDER**

1. Se crea primero la base de datos en MYSQL con el nombre de frutería\_master con cotejamiento utf8mb4\_general\_ci. Luego se crea una tabla llamada frutas y se insertan dos registros.
2. Conectamos Laravel a nuestra base de datos y para ello se hacen unos ajustes en el archivo aprendiendo-laravel/.env Y esta configuración la toma laravel en el archivo aprendiendo-laravel\config\databse.php
3. Ahora aprenderemos a trabaja con la base de datos con Laravel y vamos a trabar primero con las migraciones las cuales me permiten manejar las versiones de la misma. Con ello podemos pasar el proyecto a otra persona y que ella vea las diferentes versiones que se han manejado. Estas migraciones no son estrictamente necesarias.

Para crea migraciones se hace mediante el siguiente comando:

php artisan make:migration create\_usuarios\_table

El comando anterior tiene los componentes de:

Nombre de la migración = create\_usuarios\_table

Si queremos que nos genere los elementos por ejemplo le decimos que cree latabla mediante = --table=usuarios

Las migraciones se guardan en: aprendiendo-laravel\database\migrations

Y as migraciones se compoen de:

1. Los import que se requieren (use …). Blueprint nos permite crear los campos de la tabla a través de POO. Migration permite definir los métodos básicos de la migración, por ejemplo up me permite ejecutar la migración y con down echo para atrás la ejecución. Schema nos permite crear la tabla en si.
2. Para ejecutar las migraciones se hace mediante el siguiente comando

php artisan migrate

1. Para revertir los comandos de la migración se realiza mediante el comando:

php artisan migrate:rollback

1. Si realizamos un cambio en una migración por ejemplo adicionar un campo a una tabla y queremos adicionar es campo debemos de refrescar las migraciones para que las borre y las vuleva a ejecutar y ello se logra mediante el comando:

php artisan migrate:refresh

**PROYECTO LARAVEL (SIMILAR A INSTAGRAM)**

1. Para la creación del proyecto se realiza la ejecución del siguiente comando:

composer create-project "laravel/laravel:^9.0" instagram-laravel --prefer-dist

1. Luego de ello se crea la Base de datos, en este caso se creo con el nombre de Instagram\_master y todo el SQL se dejo en el archivo database.sql del proyecto.
2. Se ajusta el archivo .env para definir allí la URL del proyecto y se cambia el nombre de la ase de datos.
3. Ahora vamos a trabajar con Elocuent para poder trabajar los modelos que son importantes en la arquitectura MVC. En la creación de modelos, ya Laravel trae por defecto un modelo que es el de Usuario y con sus métodos.

Lo que debemos de tener en cuenta es que los modelos son creados en Singular porque representan a un solo elemento.

Lo que se hace acá es crear los otros modelos que se requieren en este caso para Images, Comments y Likes. Para ello por consola se realiza dicha creación por el comando:

php artisan make:model Image

php artisan make:model Comment

php artisan make:model Like

1. Ahora procedemos a configurar las entidades y sus relaciones. En app\models allí en cada una de las entidades se les definieron las relaciones.
2. Pasamos de nuevo al archivo de database.sql y allí creamos los string para insertar unos registros a las tablas.
3. Luego realizamos la prueba con ORM el cual nos permite que nos ahorremos desarrollando consultas de SQL, por ejemplo de mostrar las imágenes y luego tomar el id para sacar otra consulta y mostrar los comentarios y luego sacar el id de la imagen y otra vez consultar cuantos like tiene. Todo esto se puede hacer de la siguiente manera:

// Para probar el ORM hacemos uso de la clase ne este caso image

Use App\Models\Image;

Route::get('/', function () {

    // Con esto traemos todas las imagenes que tenemos en la base de datos

    $images = Image::all();

    foreach($images as $image) {

        //var\_dump($image);

        echo $image->image\_path."<br>";

        echo $image->description."<br>";

        // Si yo quiero sacar el usuario que ha creado esta imagen

        echo $image->user->name.' '.$image->user->surname."<br>";

        // Tambien quiero mostrar los comentarios que ha tenido esa imagen

        if (count($image->comments) >= 1) {

            echo '<h4>Comentarios</h4>';

            foreach($image->comments as $comment){

                echo $comment->user->name.' '.$comment->user->surname.": ";

                echo $comment->content."<br>";

            }

        }

        // Tambien podriamos sacar la cantidad de likes.

        echo 'LIKES: '.count($image->likes);

        echo "<hr>";

    }

    die();

    return view('welcome');

});

Lo que hace ORM es convertir todo en objetos y así se manejan todos anidados.

1. Ahora se realiza la Autenticación y Registro de Usuarios y esto es sencillo porque Laravel ya tiene unos métodos y toda la funcionalidad y vistas que ya Laravel las ha realizado.

Para hacer uso en versiones superiores la 5 de Laravel se realiza a través de Breeze. Para poder instalar ese paquete se realiza mediante el siguiente comando:

composer require laravel/breeze:^1.9

Luego para decirle al proyecto que haga uso de ese paquete debemos de instalarlo mediante el siguiente comando:

php artisan breeze:install

Instalado el paquete de breeze realizamos la instalación de estos módulos de npm y node con el comando:

npm install && npm run dev

Luego ejecutamos las migraciones para actualizar la base de datos:

php artisan migrate

Si ya se tiene una tabla de users es probable que salga un error porque en las migraciones se va a crear de nuevo esta tabla. Para tener en cuenta que si no desea crear de nuevo la tabla con migraciones por lo menos esta tabla debe de tener los siguientes campos:

id

name

email

email\_verified\_at (nullable)

password

remember\_token (nullable)

created\_at

updated\_at

1. Instalado Breeze me crea unas rutas en web.php y es allí donde se realiza un ajuste para cargar el login. La ruta que se tiene como welcome se quita y se deja esta ruta:

Route::get('/', function () {

    return view('dashboard');

})->middleware(['auth', 'verified'])->name('dashboard');

1. Realizamos algunos ajustes en RegisteredUserController donde se incluyen dos campos adicionales de surname y nick en cuanto a validaciones y para el método créate se envían esos datos.

En el modelo de User.php también se adicionan los nuevos campos de surname y nick.

1. En la vista de dashboard realizamos el cambio del titulo que estaba así:

{{ \_\_(‘Dashboard’) }}

Se reemplazo por

{{ \_\_(‘Larafoto) }}

1. Cambiamos el titulo de la pagina con el ajuste en el archivo .env allí en la variable de APP\_NAME.
2. Tengo pendiente el menú para colocar la opción de Subir Imágenes.
3. Para realiza la actualización del Perfil de Usuario y que también pueda realizar el cambio de los campos de Nick y Surname estos se agregan en la vista update-profile-information-form.
4. En el archivo de ProfileController realizamos unos ajustes al método de update, allí ponemos validaciones para todos los campos y en especial el de Nick y email para que no sean repetidos en la base de datos y acá se tiene en cuenta que si modifico por ejemplo el nombre en la validación de Nick o email ignore los datos de ese usuario que esta cambiando la información.
5. Se realiza una instalación de SweetAlert2 para poder manejar las alertas de una manera más elegante y no con las de Boostrap. Para ello lo que hicimos es en la vista de app.blade se adiciona el script de SweetAlert2 como un CDN así:

<!-- Scripts SweetAlert2 -->

<script src="https://cdn.jsdelivr.net/npm/sweetalert2@11"></script>

1. Con este SweetAlert2 por ejemplo para cuando se actualiza el usuario se ajusto allí en la vista update-profile-information-form este código:

@if (session('status') === 'profile-updated')

<script>

Swal.fire({

icon: 'success',

title: '¡Perfil actualizado!',

text: 'Tu información fue guardada correctamente.',

confirmButtonColor: '#3085d6',

confirmButtonText: 'OK'

});

</script>

@endif

1. Ahora lo que realizamos a continuación es la de colocar otro campo para que se puedan subir imágenes y este sea un avatar del usuario. Para ello en el archivo de la vista update-profile-information-form se adiciona esto:

<div>

            <x-input-label for="image" :value="\_\_('Avatar')" />

            <x-text-input id="image" name="image" type="file" class="mt-1 block w-full" accept="image/\*" />

            <x-input-error class="mt-2" :messages="$errors->get('image')" />

 </div>

En Laravel no se puede subir la imagen a una carpeta directa en el sistema de archivos, esto se hace mediante el uso de discos virtuales para tener un poco más protegidos los archivos que subimos al servidor y más organizados.

Lo primero que se hace es configurar el filesystem, para ello ingresamos a app\config\filesystem.php y es acá donde puedo crear los discos virtuales para guardar mis datos. Uno de los discos ya creados acá es public donde se almacenan nuestras imágenes. Se crea uno de users para guardar las imágenes de los usuarios. Y por ultimo se crea un disco de images que es donde colocaremos las imágenes de la plataforma.

Otro ajuste que se hace es en el formulario de actualización es colocar allí el enctype="multipart/form-data" para poder recibir archivos en el formulario.