Proyecto de Software

Trabajo Final – Desarrollo del trabajo de la cursada usando un framework PHP

Alumno: Prince, Diego Abel

1. Fundamentos de la elección

Laravel es un framework de código abierto que intenta aprovechar las ventajas de otros frameworks como Ruby on Rails, Sinatra y ASP.NET para desarrollar aplicaciones con código PHP de forma elegante y simple basado en un modelo MVC [1].

Laravel es un framework joven con gran futuro. Cuenta con una comunidad llena de energía, documentación atractiva de contenido claro y completo; y, además, ofrece las funcionalidades necesarias para desarrollar aplicaciones modernas de manera fácil y segura [2].

Razones para elegir Laravel [3]:

1) Técnica de autorización:
Laravel hace que la implementación de técnicas de autenticación sea muy simple. Casi todo está configurado extraordinariamente. Laravel también proporciona una forma sencilla de organizar la lógica de autorización y controlar el acceso a los recursos.

2) Bibliotecas Orientadas a Objetos:
Una de las principales razones que hacen de Laravel el mejor framework de PHP es que tiene bibliotecas orientadas a objetos y muchas otras preinstaladas, que no se encuentran en ningún otro framework de PHP popular.

3) Artisan:
Por lo general, un desarrollador tiene que interactuar con Laravel utilizando una línea de comandos que crea y maneja el entorno del proyecto Laravel. Laravel proporciona una herramienta integrada para la línea de comandos llamada Artisan. Esta herramienta nos permite realizar la mayoría de esas tareas de programación, repetitivas y tediosas que la mayoría de los desarrolladores evitan realizar manualmente.

4) Soporte MVC:
Laravel admite la arquitectura MVC, lo que garantiza la claridad entre la lógica y la presentación. MVC ayuda a mejorar el rendimiento, permite una mejor documentación y tiene múltiples funcionalidades integradas. Así es como funciona el MVC para Laravel.

5) Seguridad:
Laravel cuida la seguridad dentro de su framework. Utiliza una contraseña con hash, lo que significa que la contraseña nunca se guardaría como texto sin formato en la base de datos. Utiliza el algoritmo de hash de Bcrypt para generar una representación cifrada de una contraseña. Laravel utiliza sentencias SQL preparadas que hacen que los ataques de inyección sean inimaginables. Junto con esto, Laravel proporciona una forma sencilla de evitar la entrada del usuario para evitar la inyección de la etiqueta <script>.

Características de seguridad que ofrece Laravel: Configuración, Almacenamiento de contraseñas, Autenticación de usuarios, Protección de rutas, Encriptación, Recuperación de contraseñas.

6) Migración de la base de datos
Un punto difícil para los desarrolladores es mantener la base de datos sincronizada entre las máquinas de desarrollo. Con la migración de base de datos de Laravel, es extremadamente fácil. Mientras mantenga todo el trabajo de la base de datos en migraciones y semillas, usted puede migrar fácilmente los cambios a cualquier otra máquina de desarrollo que tenga. Esta es otra razón más que hace de Laravel el mejor framework PHP.

7) Laracasts:
Laravel ofrece Laracasts, una combinación de tutoriales en video gratuitos y de pago que muestran cómo usar Laravel. Ofrece instrucciones claras y concisas. La calidad de la producción es alta y las lecciones están bien pensadas y son significativas.

8) Motor de plantillas Blade:
El motor de plantillas Blade de Laravel es muy intuitivo y ayuda a trabajar mucho mejor con los espaguetis típicos de PHP/HTML, es una de las mejores características del framework.

10) Descubrimiento automático de paquetes
Permite detectar los paquetes que los usuarios desean instalar automáticamente. Lo que significa que los usuarios no tienen que configurar ningún alias o proveedor para instalar nuevos paquetes en Laravel.

2. Aspectos técnicos evaluados

2.1. Mecanismo provisto para el manejo de seguridad

2.1.1. Protección contra CSRF

Laravel genera automáticamente un "token" CSRF para cada sesión de usuario activa gestionada por la aplicación. Este token se usa para verificar que el usuario autenticado es el que realiza las solicitudes a la aplicación.

Puede usar la directiva Blade “@csrf” para generar el campo del token en el formulario HTML. Adicionalmente, también puede guardarlo en una etiqueta <meta> para incluirlo manualmente en los encabezados de sus peticiones:

<meta name = "csrf-token" content = "{{csrf_token ()}}">

El middleware *VerifyCsrfToken*, que se incluye en el grupo de middlewares para peticiones web, verificará automáticamente que el token en la entrada de solicitud coincida con el token almacenado en la sesión [4].

2.1.2. Autenticación

Laravel hace que la implementación de la autenticación sea muy simple. El archivo de configuración de autenticación se encuentra en *config/auth.php*, que contiene varias opciones bien documentadas para ajustar el comportamiento de los servicios de autenticación.

Básicamente, las instalaciones de autenticación de Laravel están formadas por "guardias" y "proveedores". Los guardias definen cómo se autentican los usuarios para cada solicitud. Por ejemplo, Laravel incluye un protector de sesión que mantiene el estado mediante el almacenamiento de sesión y las cookies.

Los proveedores definen cómo se recuperan los usuarios de su almacenamiento persistente. Laravel provee soporte para recuperar usuarios usando Eloquent y el generador de consultas de base de datos. Sin embargo, es libre de definir proveedores adicionales según sea necesario para su aplicación.

Accederemos a los servicios de autenticación de Laravel a través de la fachada Auth, usando el método *Auth::attempt()* al cual se le pasa un arreglo clave/valor. Los valores del arreglo se utilizarán para encontrar al usuario en la tabla de la base de datos. El método *attempt()* iniciará la sesión para el usuario y devolverá verdadero si la autenticación fue exitosa. De lo contrario, se devolverá falso [5].

2.1.3. Protección de rutas

Los middleware se pueden usar para permitir acceder a una ruta determinada solo a los usuarios que cumplen ciertas condiciones, como puede ser el estar correctamente autenticado. Un forma de asignar un middleware a una o varias rutas es llamándolo desde el constructor de un controlador y vinculándolo al método que responde a dicha ruta [6].

Entonces, por ejemplo para asignar el middleware encargado de verificar si el usuario está correctamente autenticado (que es uno de los que viene por defecto con Laravel), dentro del constructor del controlador se debe poner:

```
$this->middleware('auth');
```

Si se quiere solo para algunos métodos particulares:

```
$this->middleware('auth')->only(['verUsuarios', 'crearUsuario']);
```

2.1.4. Protección contra XSS

Por defecto el motor de plantillas Blade filtra todas las salidas que se producen en el HTML, adicionalmente como segunda barrera se puede crear un middleware global que intercepte las peticiones para sanitizar sus campos.

2.1.5. Protección contra inyección SQL

Laravel provee la fachada “DB”, la cual posee métodos para realizar consultas con vinculación de parámetros que otorga protección contra inyecciones SQL.

2.2. Mecanismo provisto para el manejo de rutas

Todas las rutas de Laravel se definen en sus archivos de ruta, que se encuentran en el directorio *routes*. Estos archivos son cargados automáticamente por el framework. El archivo *route/web.php* define las rutas que son para su interfaz web. A estas rutas se les asigna el grupo *web* de los middleware, que proporciona funciones como el estado de la sesión y la protección CSRF. Las rutas en *route/api.php* son sin estado y están asignadas al grupo *api* de los middleware.

Las rutas definidas en el archivo route/api.php están anidadas dentro de un grupo de rutas por RouteServiceProvider. Dentro de este grupo, el prefijo */api* se aplica automáticamente, por lo que no es necesario que lo aplique manualmente a todas las rutas del archivo.

La forma de definir una ruta es la siguiente:

Route::<method1>('<uri>', '<Controller>@<method2>')

Donde <method1> puede ser get, post, put, patch, delete u options.
Donde <uri> es un string para identificar la ruta que puede contener parámetros que se pasan al método del controlador, por ejemplo 'user/{id}'.
Donde <Controller> es el nombre de la clase del controlador.
Donde <method2> es el nombre del método que se va a invocar en el controlador.

Laravel provee más funciones que enriquecen el manejo de rutas [7], como por ejemplo la definición de una respuesta para rutas que no existan, redirecciones, grupo de rutas, rutas que responden directamente con una vista, etc.

2.3. Manejo de MVC: Árbol de directorios

La estructura de Laravel [8] se organiza de la siguiente forma:

2.3.1. El directorio Root

2.3.1.1. El directorio App

Contiene el código central de su aplicación. Exploraremos este directorio en el punto 2.3.2

2.3.1.2. El directorio Bootstrap

Contiene el archivo app.php que inicializa el framework. Este directorio también alberga un directorio de caché que contiene archivos generados por el framework para la optimización del rendimiento, como los archivos de caché de rutas y servicios.

2.3.1.3. El directorio Config

Contiene todos los archivos de configuración de su aplicación.

2.3.1.4. El directorio Database

Contiene las migraciones de su base de datos, las fábricas de modelos y las semillas. Si lo desea, también puede usar este directorio para mantener una base de datos SQLite.

2.3.1.5. El directorio Public

Contiene el archivo index.php, que es el punto de entrada para todas las solicitudes que ingresan a su aplicación y configura la carga automática. Este directorio también alberga sus recursos, como imágenes, JavaScript y CSS.

2.3.1.6. El directorio Resources

Contiene sus vistas, así como sus recursos sin compilar y sin procesar, como LESS, SASS o JavaScript. Este directorio también contiene todos sus archivos de idioma.

2.3.1.7. El directorio Routes

Contiene todas las definiciones de ruta para su aplicación. De forma predeterminada, se incluyen varios archivos de ruta con Laravel: *web.php*, *api.php*, *console.php* y *channels.php*.

2.3.1.8. El directorio Storage

Contiene sus plantillas Blade compiladas, sesiones basadas en archivos, cachés de archivos y otros archivos generados por el framework.

2.3.1.9. El directorio Tests

Contiene sus pruebas automatizadas.

2.3.1.10. El directorio Vendor

Contiene sus dependencias de Composer.

2.3.2. El directorio App

2.3.2.1. El directorio Broadcasting

Contiene todas las clases de canales de transmisión para su aplicación. Estas clases se generan utilizando el comando make: channel.

2.3.2.2. El directorio Console

Contiene todos los comandos personalizados de Artisan para su aplicación. Estos comandos se pueden generar usando el comando make: command. Este directorio también alberga el núcleo de su consola, que es donde se registran sus comandos personalizados de Artisan y se definen sus tareas programadas.

2.3.2.3. El directorio Events

Contiene clases de eventos. Los eventos pueden usarse para alertar a otras partes de su aplicación de que se ha producido una acción determinada, proporcionando una gran flexibilidad y desacoplamiento.

2.3.2.4. El directorio Exceptions

Contiene el controlador de excepciones de su aplicación y también es un buen lugar para colocar cualquier excepción lanzada por su aplicación.

2.3.2.5. El directorio Http

Contiene sus controladores, middleware y solicitudes de formularios. Casi toda la lógica para manejar las solicitudes que ingresan a su aplicación se ubicará en este directorio.

2.3.2.6. El directorio Jobs

Contiene los trabajos en cola para su aplicación. Su aplicación puede poner en cola los trabajos o ejecutarse de forma síncrona dentro del ciclo de vida de la solicitud actual. Los trabajos que se ejecutan sincrónicamente durante la solicitud actual a veces se denominan "comandos", ya que son una implementación del patrón de comandos.

2.3.2.7. El directorio Listeners

Contiene las clases que manejan tus eventos. Los oyentes de eventos reciben una instancia de evento y realizan una lógica en respuesta al evento que se está activando.

2.3.2.8. El directorio Mail

Contiene todas sus clases que representan correos electrónicos enviados por su aplicación. Los objetos de correo le permiten encapsular toda la lógica de crear un correo electrónico en una sola clase simple que puede enviarse usando el método Mail::send.

2.3.2.9. El directorio Notifications

Contiene todas las notificaciones "transaccionales" que envía su aplicación, como notificaciones simples sobre eventos que ocurren dentro de su aplicación. Las funciones de notificación de Laravel resumen el envío de notificaciones a través de una variedad de controladores como correo electrónico, Slack, SMS o almacenados en una base de datos.

2.3.2.10. El directorio Policies

Contiene las clases de políticas de autorización para su aplicación. Las políticas se utilizan para determinar si un usuario puede realizar una acción determinada contra un recurso.

2.3.2.11. El directorio Providers

Contiene todos los proveedores de servicios para su aplicación. Los proveedores de servicios inician el arranque de su aplicación vinculando los servicios en el contenedor del servicio, registrando eventos o realizando cualquier otra tarea para preparar su aplicación para solicitudes entrantes.

2.3.2.12. El directorio Rules

Contiene los objetos de regla de validación personalizados para su aplicación. Las reglas se utilizan para encapsular una lógica de validación complicada en un objeto simple.

2.3.3. Directorio de modelos

Laravel encuentra la palabra “modelos” ambigua, por este motivo, elige ubicar los modelos de Eloquent en el directorio de la aplicación de manera predeterminada, y permitir que el desarrollador los coloque en otro lugar si así lo desea.

2.3.4. Conclusión

Por lo tanto las clases del modelo se ubicaran en una carpeta “Model” dentro del directorio App (elección del desarrollador), por otro lado las vistas se ubicaran en el directorio /app/resources/views y por último los controladores se ubicaran en /app/http/controllers.

3. Migración del trabajo de cursada al framework

Se comenzó migrando la vista, se migro la implementación en Twig a su similar en Blade. Dichas implementaciones no difieren mucho, por lo cual el proceso fue simple y rápido. Por otro lado, el mecanismo para seleccionar que vista mostrar desde el controlador ya viene incorporado con el framework, por lo cual no fue necesario hacer clases que se encarguen de mostrar las vistas (procesar los templates).

El manejo de rutas que se hacía en index.php se migro a routes/web.php, cabe destacar que los controles que se hacían en index.php se migraron a los middlewares correspondientes y además ya no es necesario pasar una variable “action” para indicar la ruta que se desea acceder. Se aprovechó para agrupar rutas y mejorar su legibilidad.

En cuanto a la migración de los controladores, se simplifico la comunicación con el modelo gracias a la inyección de dependencias que provee Laravel, se utilizó el estándar que propone el framework acerca de las operaciones CRUD para nombrar funciones y rutas, también se utilizaron los mecanismos de validación de datos provistos por Laravel. Todo esto permitió simplificar el código haciéndolo más expresivo y legible.

Por otro lado, en cuanto a las sesiones, el framework brinda una serie de funciones para su manejo para así no tener que operar directamente con la variable $_SESSION. En cuanto a los casos en los que se necesita devolver los datos en formato JSON, Laravel los convierte directamente si se retorna un arreglo, por lo tanto no es necesario implementar una vista que se encargue de esto o usar funciones adicionales.

Como se dijo anteriormente, en los controladores se utilizan middlewares los cuales realizan los controles que se hacían en index.php.

El manejo de errores se migro a la clase app/Exceptions/Handler.php del framework.

En cuanto a la migración de la API REST, no se utilizó Slim sino el mecanismo que provee Laravel. Por lo tanto, la implementación del ruteo se hizo en routes/api.php y se creó un controlador ApiController donde se migro la funcionalidad. Dicho controlador es similar a los nombrados anteriormente, utiliza inyección de dependencias para la comunicación con el modelo y middlewares para el control de autenticación y autorización. El control de autorización y autenticación es un agregado ya que se debe extender el trabajo con funcionalidad para dar de alta y modificar instituciones, para este control se utiliza Passport.

Por último, en cuanto a la migración del modelo, se delegó la conexión al framework permitiendo así descartar el archivo conexión.php y su inclusión en cada clase del modelo. Se reemplazó la utilización de PDO por el constructor de consultas provisto por Laravel para realizar las consultas contra la base de datos.

4. Conclusiones

Laravel es sin duda un framework muy completo que una vez aprendido, puede ser utilizado para un rápido desarrollo de una aplicación, manteniendo prolijidad y eficiencia.

Es sencillo de instalar debido a que usa composer y a su vez fácil de configurar, gracias a sus archivos de configuración y a los comandos de Artisan.

Ofrece varias alternativas para implementar sus diferentes aspectos, como por ejemplo el manejo el acceso a datos, que puede hacerse con un ORM o con un constructor de consultas.

Su sistema de inyección de dependencias es completo e intuitivo, sin complejidades.

Posee una documentación amplia y descriptiva además de una gran comunidad base, lo que es importante a la hora de buscar material acerca del framework o soluciones a problemas que se producen en el mismo.

Lo que más destaco durante el desarrollo fue la simplicidad que ofrece para el manejo de rutas y la protección de estas mediante middlewares.

5. Referencias

[1] www.laravel.com

[2] https://www.ecured.cu/Laravel#Caracter.C3.ADsticas

[3] https://www.valuecoders.com/blog/technology-and-apps/laravel-best-php-framework-2017/

[4] https://laravel.com/docs/5.8/csrf

[5] https://laravel.com/docs/5.8/authentication

[6] https://laravel.com/docs/5.8/middleware

[7] https://laravel.com/docs/5.8/routing

[8] https://laravel.com/docs/5.8/structure