Informe final de promoción
Proyecto de Software 2016

Autor: Minetto Fermín

1) Fundamentación del framework elegido

El framework elegido para hacer la promoción fue Laravel. En la clase de proyecto vi que fue el más votado en GitHub y por otra encuesta realizada a desarrolladores con amplia diferencia de Symfony. Por esa razón inicial decidí comenzar a estudiar para ver si me gustaba. A medida que fui viendo videos de Laravel me fui dando cuenta que era un gran framework (que tomó cosas de otros framework, muchas de ellas ya pulidas gracias a los años de las comunidades) y que podría utilizar la mayoría de los componentes que ofrecía para facilitar la tarea de desarrollo enormemente reduciendo el código a escribir por uno mismo en cosas que son comunes a muchas aplicaciones.
También investigué un poco de Symfony, que era el segundo framework más utilizado por los desarrolladores, pero finalmente me decanté por Laravel aunque no me pareció que le faltaran cosas o resultara más indicado que Laravel para el proyecto a desarrollar.

2) Referencias y material estudiado

El framework fue aprendido de manera autodidacta utilizando distintos materiales encontrados por la web. Casi la totalidad del framework fue aprendido leyendo la completísima documentación oficial de Laravel; cargada de ejemplos para el uso de cada componente, que también se traduce en guías de buenas prácticas de programación y uso de sintaxis en Laravel (que tiene su propía "filosofía" en cuanto a sintaxis).
Una pequeña parte de la curva de aprendizaje (que nunca se acaba) realizada hasta este momento se realizó gracias a la ayuda de un conocido y renombrado canal de YouTube llamado *codigofacilito,* en la cual un experto desarrollador en Laravel (Uriel Hernández) explica técnicas y los componentes escenciales de Laravel. Este canal de YouTube sirvió para dar los primeros pasos introductorios en Laravel y conocer sus componentes principales, que pueden ser un poco intimidantes al principio, antes de ponerme de lleno con la documentación oficial de Laravel.

3) Descripción de modulos de la cursada reaprovechados

Se volvió a escribir la mayor parte del código puesto que Laravel siempre proveía componentes que facilitaban en gran medida las tareas que teníamos que implementar. Debido a que el objetivo de aprender a manejar un framework es usar los componentes que provee y no "reinventar la rueda", decidí reemplazar el código existente con código utilizando componentes del framework.
Un módulo de cursada que fue reutilizado y refactorizado ligeramente fue el del bot,

que pasó a ser parte de un controlador propio (BotController), pero que el esqueleto de recibir, captar la orden y devolver una respuesta respetando la API de Telegram fue mantenido (aunque distribuído cuando se hizo refactoring).

4) Mecanismo provisto para manejo de seguridad y routing

Vulnerabilidades de seguridad (más comunes, no todas, se tienen en cuenta muchas más pero por cuestiones de practicidad no son descriptas) que contempla el sistema:

Injection: Top 1 en la OWASP. Eloquent parametriza las consultas para evitar una inyección de SQL proveniente de datos inseguros. Los tests de inyección fueron superados y no se pudo hacer una inyección de SQL.
Pobre manejo de sesiones: Top 2 en la OWASP, las sesiones se mantienen respetando standards. Las credenciales de los usuarios se encuentran almacenadas de forma segura y no se implementó un "sistema de autenticación y manejo de sesiones casero" sino que se utilizó el provisto por el framework.
Inyección XSS: La vulnerabilidad top 3 en la OWASP no puede realizarse en SysBuffet puesto que los datos que se "imprimen" por medio del gestor de plantillas (todos los datos que se les pasa a la vista) son escapados por éste último.
CSRF: La vulnerabilidad top 7 en la OWASP se encuentra contemplada. Laravel implementa un sistema de manejo de tokens CSRF que son gestionados de manera casi automática (de forma completamente automática si se utiliza el componente extra *laravel-forms* ya que añade el input hidden del token de manera automática).

El tema del routing en específico está descripto en la parte MVC. En cuanto a la protección de rutas se usan ***middlewares***, que serían una especie de "filtro" HTTP, que se ejecutan antes de pasar un requerimiento HTTP a la ruta correspondiente. En este filtro se puede decidir si continuar con el requerimiento (por ejemplo, si es que el usuario esta logeado o posee el rol adecuado) o no brindarle acceso. Aunque este filtro podría tener otras utilidades (por ejemplo, un contador de visitas algo rústico), **en el sistema fue utilizado para proveer la seguridad de la protección de las rutas y accesos indebidos**.

5) Mecanismo provisto para operaciones CRUD

La arquitectura CRUD del sistema en su mayor parte está compuesta por listas que agrupan los recursos de interés (ejemplo: productos, usuarios, compras, etc). En esos listados se encuentra un mensaje a la API (naturalmente, dependiendo del recurso, este variará) para borrarlos. En caso de querer modificarlos o darlos de alta se podrá pasar a ventanas de modificación/alta.
Esta "arquitectura" se sustenta fuertemente en la ***reutilización del código de las vistas*** (las cuales son manejadas por el gestor de plantillas ***Blade***, incluído en Laravel), proveyendo de un esqueleto de listados y altas/bajas en que solo se modifican los propios formularios, que estos si variarán de acuerdo al recurso utilizado.

El uso de formularios se administra con un componente que solía incluirse con Laravel pero que fue dejado aparte para reducir el tamaño total del framework. Este componente se llama ***laravel-forms*** y es aconsejado su uso debido a un gran abanico de funcionalidades que posee para flexibilizar el uso de formularios, mensajes PUT y DELETE (que de otra manera no vienen soportados por HTML5) y asociaciones de datos del formulario con los modelos.

Del lado propio del modelo, se utiliza ***el mapeador para persistir los objetos del modelo*** que viene incluído en ***Laravel***: el Eloquent. Para la generación de las tablas en la base de datos se utiliza un componenete llamado ***migraciones*** de Laravel, que nos permite definir la estructura de las tablas para poder generarlas ejecutando un simple comando de *artisan*, la consola de Laravel.

6) Forma de manejar el MVC

El modelo, como se describió arriba, está compuesto por las clases del sistema que son persistidas por el componente ***Eloquent*** de Laravel que se encarga de persistir estos objetos en la base de datos y resolver consultas sin necesidad de "casarse" con un gestor de base de datos específico, proveyendo una abstracción muy útil en cuanto a la parte de persistencia de objetos.

Los controladores **es una clase** provista por Laravel, que nos permite asociar sus mensajes a filtros por defecto (*middlewares*) para proveer seguridad o alguna funcionalidad extra, y para **mapear las rutas con mensajes del controlador.** El ruteo en Laravel se realiza en un archivo, anotando las rutas que entiende nuestro sistema, o dando directivas para mapearlas automáticamente (por ejemplo, para que si tenemos un recurso con un CRUD típico como un producto, se comprenda que las rutas get, post, put y delete estén mapeadas a mensajes que pueden ser auto-generados en el controlador) y así **ahorrar espacio en el archivo de rutas.** Los controladores del sistema llaman a mensajes del modelo donde se ejecuta la lógica de negocio y luego devuelven vistas.

Las vistas de Laravel, al igual que todos los componentes del framework, se encuentran en una carpeta específica que puede ser a su vez dividida en subcarpetas para mantener un orden. **Las vistas son gestionadas por el gestor de plantillas Blade**, que viene incluído en Laravel. Es muy parecido al Twig, que se usó en las primeras fases del trabajo.