

# Informe de Trabajo

Proyecto de Software

## Integrantes:

- Barbá, Dante
- Loscalzo, Jonathan

Framework: Laravel

## Fundamentos para la elección.

### ¿Por qué Laravel?

Luego de realizar tablas comparativas de frameworks, hemos elegido a Laravel como la mejor opción gracias a su popularidad, simplicidad, facilidad de aprendizaje, documentación y recursos, como por ejemplo Laracast. Si bien otros frameworks tenían mejor documentación, como por ejemplo Symfony, la popularidad de Laravel lo hace ideal para el aprendizaje en un corto período de tiempo. Todos los comparados son proyectos Open-Source.

Laravel es un framework diferente, su sintaxis lo hace distinto y que sobresalga frente a otros frameworks. Laravel trata de sacar jugo a las nuevas versiones de Php, aprovechando el manejo de objetos, patrones, UnitTesting, aunque manteniendo un poco menos de compatibilidad con versiones viejas del lenguaje ( lo cual no es un problema ). Comparado con otros frameworks, el manejo de rutas hace recordar a Sinatra de Ruby y el uso del patrón ActiveRecord a Rails.

*“Laravel es accesible, pero potente, ofreciendo herramientas poderosas necesarias para aplicaciones de gran envergadura. Un espléndido contenedor de inversión de control, sistema de migraciones expresivo, y un soporte para pruebas unitarias estrechamente integrado, te entregan las herramientas que necesitas para construir cualquier aplicación que te hayan encargado.”*

## Comparación con otros Frameworks PHP

### CodeIgniter

Si bien CodeIgniter es un framework sencillo, su funcionalidad es más limitada si no se agregan módulos extra. En cuanto a performance es muy bueno, pero no lo consideramos un punto necesario para este trabajo. Con referencia a la base de datos, CodeIgniter tiene un query builder funcional, pero difícil de utilizar y además, la forma de utilizar los modelos es un poco “engorrosa” por la falta de un verdadero patrón como ActiveRecord y el Façade utilizado en Laravel, de esta manera, “Convention Over Configuration” facilita el uso y las migraciones son un punto fuerte a la hora de desarrollo ágil. CodeIgniter no incentiva el uso de templates (aunque no significa que no se puedan utilizar), en cambio Laravel tiene por defecto el uso de Blade. Todo esto, genera que una aplicación Laravel sea más legible, por el mismo funcionamiento del framework, en cambio con CodeIgniter deberían agregarse módulos extras, con el riesgo que esto conlleva (distintas comunidades, compatibilidad, etc).

## Zend2

Zend2 en un principio fue una alternativa válida, sin embargo a pesar de su documentación accesible, no se encuentra mucha información de usuarios utilizando el framework. Tampoco ofrece ninguna ventaja extra comparado con Laravel. Las rutas nos pareció más fácil en Laravel, y la forma de crear código automático con Artisan para generar RestFul nos pareció aún más legible. También en este caso, al igual que CodeIgniter, no se utiliza el patrón ActiveRecord, generando un punto de ventaja y legibilidad, al igual que desarrollo ágil para Laravel. En conclusión, es una herramienta similar a CodeIgniter, donde tu configuras tu framework a medida, en cambio Laravel, es más parecido al estilo de Symfony, un poco más automatizado. CodeIgniter y Zend son para desarrolladores con mayor experiencia en desarrollo web.

## Yii

Comparado con Laravel, aporta una gran facilidad para la validación de formas y las operaciones CRUD, pero la discontinuación de la versión, ciertas quejas con respecto a la documentación, la falta de usuarios activos y ciertos problemas de compatibilidad nos llevaron a abandonar la idea. La comunidad en Github, es mucho más activa con Laravel que con Yii2.

## Symfony

Finalmente quedó comparar la opción Symfony con Laravel. Laravel y Symfony comparten módulos y son muy parecidos en varios aspectos. Con Symfony existía la ventaja de utilizar Twig, sin necesidad de migrar todas las plantillas a Blade, el sistema de plantillas de Laravel. Symfony2 facilita las tareas habituales, que cubrirían pequeños desarrollos web, permitiendo trabajar a bajo nivel con componentes reutilizables. Muchos desarrollos utilizan symfony, incluso el mismo Laravel.

Sin embargo conocíamos que en la facultad se hacían cursos de Symfony2, que la curva de aprendizaje era mayor y que la cantidad de usuarios era inferior a Laravel 4. Decidimos arriesgarnos a una opción nueva, para aprender un framework que no se utiliza activamente en los cursos y cursadas de la facultad y que tiene una de las mayores cuotas del mercado.

## ¿Por qué Laravel 4?

Es de público conocimiento la existencia de una versión nueva de Laravel llamada Laravel 5. La razón primordial por la cual no decidimos utilizar la versión más actualizada es debido a que se encuentra en fase de desarrollo (versión Alpha) y

la documentación existente es muy escueta. Al ser novatos con el framework utilizar la versión mas estable y mejor documentada es la mejor alternativa. Además, cuando comenzamos el desarrollo aún no había salido oficialmente la versión 5.

## Módulos reutilizados

Para el módulo de impresiones, la traducción a Blade no costó mucho. El código JavaScript quedó prácticamente igual, con algunos cambios, la vista fue migrada con los helpers, de una manera fácil quedando código con una legibilidad mucho mayor que con Twig. Migrar los modelos para esta sección fue distinta al módulo de Login y Gestión de usuarios, debido a que nosotros habíamos utilizado vistas y procedimientos almacenados, por lo tanto tuvimos que utilizar el Query Builder que el framework provee para realizar consultas directamente. En su momento, se realizó de esta manera, porque era más fácil realizar dichas consultas en la base de datos y corregirlas, además de que no poseíamos un ORM que nos facilitara la resolución, que tenía una mayor dificultad de una simple consulta. Agregar DomPDF como un Servicio fue mucho más fácil que la instalación manual, Laravel lo configuró por nosotros y solo tuvimos que agregar el “alias” correspondiente. La lógica del controlador no cambió, si utilizamos características del framework, como el objeto Response para retornar un Json, o el objeto Input para los datos de entrada enviados desde la vista. Además descubrimos un bug en la aplicación vieja, en la que olvidábamos verificar el usuario autenticado en este módulo, que gracias a los filtros terminó siendo mucho más legible.

Para los módulos de Gestión de usuarios y Login, se utilizó la lógica, como por ejemplo validar, enviar mensajes de error, o realizar la operación requerida; pero se tuvo que migrar prácticamente todo el código, para utilizar la nueva funcionalidad que nos ofrecía el framework, como el objeto Validator, así como también el objeto Auth. Para estas vistas, solo se rehusó prácticamente todo, solo agregando el token al formulario para la vista de Gestión, y para el login fue una mezcla entre los helpers y el código HTML que ya teníamos.

## Routing

El routing en Laravel se realiza en el archivo Php especial llamado “routes.php”. En este archivo se declaran a través de una serie de clases el tipo de ruta que se desea y todos los parámetros asociados a la ruta. Route es un objeto que utiliza el patrón Façade, a través del contenedor IoC del framework. Las rutas más simples, son

similares a Sinatra de Ruby, consiste en un llamado a una URI, con un determinado Verbo HTTP y la ejecución de un closure.

```
Route::get('/', function() { return 'Hello World'; });
```

También se pueden utilizar parámetros de ruta:

```
Route::get('/api/{param}', function() { return 'Hello World'; });  
Route::get('/api/{param?}', function() { return 'Hello World'; });  
Route::get('/api/{param?}', function(param='value'){return 'Hello World'; });
```

Se puede cambiar el closure, por la acción de un determinado controlador:

```
Route::get('/route', 'controller@method');
```

Se puede agregar el uso de filtros (after o before) para determinadas rutas, o grupo de rutas. Los filtros deben colocarse en el archivo filters.php. Es uno de los métodos que mas utilizamos en el trabajo y de los que mas código ha simplificado ideal para permisos y roles:

```
Route::filter('old', function() {  
    if (Input::get('age') < 200) { return Redirect::to('home'); }  
});  
  
Route::get('user', array(  
    'before' => 'old',  
    'uses' => 'UserController@showProfile'  
));
```

Se puede agregar a las rutas un “alias” o nombre, para que sea luego más fácil o legible utilizarlo para crear un link. Se tiene que agregar el parámetro “as”:

```
Route::get('user/profile', array('as' => 'profile', function() { // }));  
y luego utilizarse así:  
$url = URL::route('profile');
```

Se pueden agrupar rutas, agregando filtros o prefijos de ruta:

```
Route::group(array('before' => 'auth'), function() {  
    //Todas las rutas creadas aquí, deben tener el filtro “auth”, es decir el usuario debe estar loggeado.  
});
```

Gracias al contenedor IoC, se puede inyectar una instancia de un modelo en una determinada ruta. De esta manera, cuando profile/1, se busca el User con id igual a 1,

y se lo inyecta en la variable \$user:

```
Route::model('user', 'User');  
Route::get('profile/{user}', function(User $user){});
```

## Seguridad

En cuanto a seguridad, Laravel proporciona mecanismos built-in para protegerse de las amenazas mas comunes, así como también una implementación sencilla para la autenticación.

## Autenticación

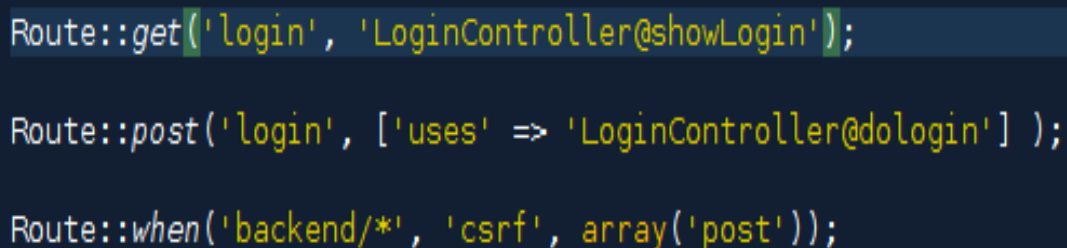
Para almacenar las contraseñas, Laravel provee la clase Hash, con esta clase se puede crear, verificar o rehashear una contraseña de un usuario.

Con la clase Auth se hace el manejo de usuarios, se puede verificar si la sesión está iniciada, desloguearse, la opción “remember me”, etc. Se puede realizar la autenticación automáticamente con el método attempt(), o manualmente con login() el cual recibe una instancia de la clase User.

Para proteger las rutas, hay que utilizar el filtro Auth.

## CSRF

Para lidiar con los ataques CSRF se utiliza un token y un filtro pre-establecido. El filtro es aplicado sobre un grupo de rutas pre-establecidas que son susceptibles a ataques. Si el token no es válido se eleva una excepción por defecto “TokenMismatch”.



```
Route::get('login', 'LoginController@showLogin');  
  
Route::post('login', ['uses' => 'LoginController@doLogin'] );  
  
Route::when('backend/*', 'csrf', array('post'));
```

*Ilustración 1: Protección CSRF en Route::when, todas las rutas /backend/\* están cubiertas.*

```
<div id="dialog" class="error" style="border: 1px solid red; padding: 5px; text-align: center; color: red; font-weight: bold; margin-bottom: 10px;">
    <p>LAS CONTRASEÑAS DEBEN SER IGUALES!.</p>
</div>
</div>
<div class="conj-block">
    <label for="roleID" style="display: block">Rol del usuario: </label>
    <select id="roleID" required name="roleID">
        <option selected disabled hidden value=""></option>
        @foreach ($roles->all() as $rol)
            <option value="{{ $rol->roleID }}"> {{ $rol->roleuser }}</option>
        @endforeach
    </select>
</div>
<button id="submit" type="submit" name="submit" disabled> Enviar </button>
{{ Form::token() }}
</form>
```

Ilustración 2: Token en HTML. Si el token no esta presente el servidor dará error

## XSS

La protección XSS se logra con Blade al utilizar '{{{ \$var }}}'. Los triple brackets escapan los strings filtrando el contenido malicioso.

## SQL-Injection

Las inyecciones SQL son automáticamente prevenidas utilizando Eloquent, la librería ORM que incorpora Laravel. Eloquent utiliza PDO a bajo nivel para las consultas, más filtros para preparar la entrada.

## CRUD

Laravel tiene un mecanismo para no escribir código CRUD una y otra vez. Normalmente, cada entidad o modelo de nuestra aplicación tendrá en el controlador, un módulo para insertar, modificar, eliminar o listar.

Utilizando los verbos HTTP: GET, POST, PUT, DELETE se pueden realizar estas consultas lo más “automatizado” posible. Agregando en el archivo de rutas la siguiente línea de código:

```
Route::resource('photos', 'PhotosController');
```

y luego con el comando de Artisan para listar las rutas de la aplicación:

```
php artisan routes
```

Laravel va a mostrar las rutas creadas, automáticamente para el recurso Photo:

GET HEAD	photos	photos.index	PhotoController@index
GET HEAD	photos/create	photos.create	PhotoController@create
POST	photos	photos.store	PhotoController@store
GET HEAD	photos/{photos}	photos.show	PhotoController@show
GET HEAD	photos/{photos}/edit	photos.edit	PhotoController@edit
PUT	photos/{photos}	photos.update	PhotoController@update
PATCH	photos/{photos}		PhotoController@update
DELETE	photos/{photos}	photos.destroy	<a href="#">PhotoController@destroy</a>

Luego con el comando:

```
php artisan controller:make PhotosController
```

Se crea un controlador con los metodos necesarios que son index(), create(), store(), show(), edit(), update() y destroy(), los cuales tenemos que completarlos con las funciones de Eloquent.

Necesitamos el modelo Photo, para poder realizar las operaciones sobre esta entidad. En Laravel 5.0 tenemos esta opción automatizada también mediante el comando:

```
php artisan make:model PhotoModel
```

el cual crea un modelo para Photo, aunque esta opción no se encuentra en artisan de Laravel 4.2, y los anteriores comandos son ligeramente diferentes, pero realizan lo mismo.

De esta manera, se automatiza la creación de código repetido, y el desarrollador solo debe centrarse en el desarrollo de la lógica de la aplicación, respetando de alguna manera REST y el Patrón MVC lo más “limpio” posible.

## MVC y árbol de directorios

Laravel tiene una estructura de carpetas jerarquizada, de esta manera el desarrollador realiza las tareas de manera semi-automatizada. Laravel es, al igual que Rails, prefiriendo “Convention over Configuration”, en vez de largos archivos de configuración XML al estilo Java o .Net. Por defecto, el framework posee la siguiente estructura:

- Laravel-Project-Base
  - app
    - commands
    - config



## Proyecto de Software

- packages
  - testing
- controllers
- database
  - migrations
  - seeds
- models
- start
- storage
- test
- views
- public
- vendor

En la parte más alta del árbol, encontramos las carpetas `/app`, `/public` y `/vendor`. La carpeta `/app` contiene controladores, modelos, vistas, base de datos (migraciones), etc; donde la mayor parte del código de la aplicación reside.

En la carpeta `/public` se puede ver desde el exterior, y contiene archivos estáticos como CSS, JavaScript o imágenes.

En la carpeta `vendor`, se contienen las dependencias del framework, packages adicionales incluso el mismo framework ( Illuminate, Symfony, etc ).

Entrando en la carpeta `/app`, tenemos las carpetas `/config`, `/controllers/`, `/databases/`, `/lang`, `/models`, `/start`, `/storage`, `/test`, `/views`.

Como archivos directamente en el directorio `/app` se encuentran `filter.php` y `routes.php`, que contienen filtros y rutas respectivamente.

En `/app/controllers` se encuentran las clases controladoras que heredan de `BaseController`, las cuales proveen la lógica del negocio, interactúan con los modelos, y cargan las vistas de nuestra aplicación.

En `/app/views` se encuentran los templates de vistas, se pueden agrupar en carpetas allí dentro como `/Layout` y cada una correspondiente a un controlador.

En `/app/test` se colocan los testUnits utilizando PHPUnit y pueden ser ejecutados con la herramienta de Artisan de Laravel.

En `/app/storage` se guardan archivos temporales del Framework, tales como sesiones, cache, vistas compiladas, etc ( el directorio debe ser “writable” por el webserver ).

Este directorio es propio del framework, y el usuario no tiene nada que tocar agregar aquí.

En `/app/start` contiene ajustes custom para la herramienta Artisan.

En `/app/models` se guardan las clases que corresponden al modelo de la aplicación, normalmente heredan de Eloquent.

En `/app/lang` se encuentran los archivos de recursos por idiomas, por defecto se encuentran las validaciones y paginación pero en inglés.

En `/app/database` se encuentran las migraciones y los “data seed” o datos con los que llenar la base de datos mediante los comandos de Artisan.

En `/app/config` se encuentran todos los archivos de configuración de la aplicación. Desde directorios por defecto, hasta connection strings para la base de datos y los drivers, servicios de mail, “queues”, etc.

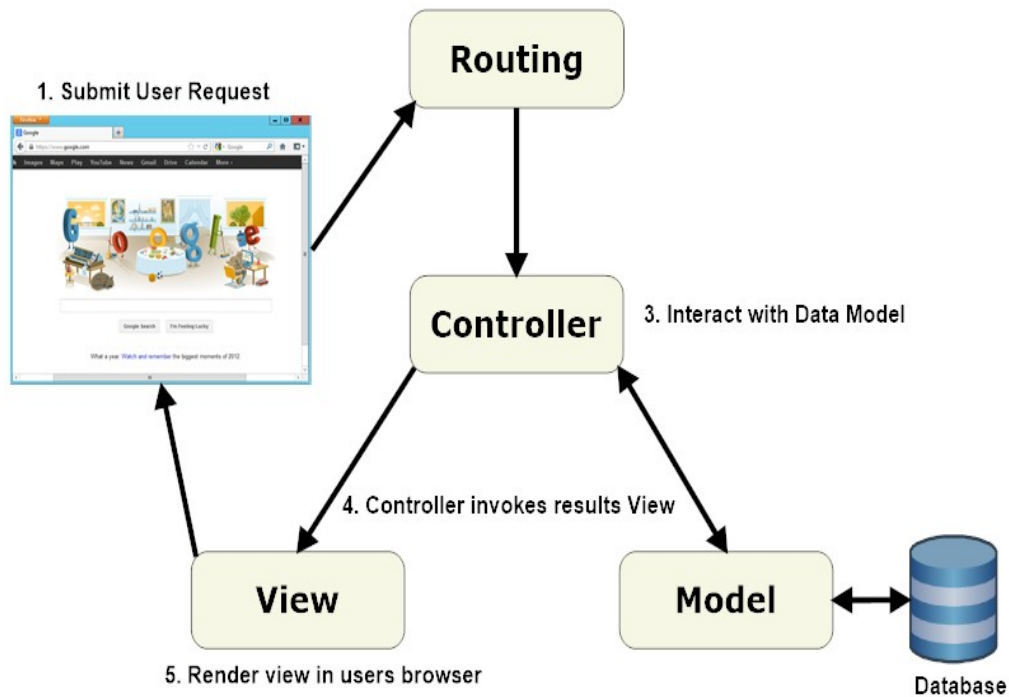
Entonces, luego de esta explicación, comenzamos a explicar el manejo de MVC de una aplicación Laravel. Como se vio en detalle, dentro de la carpeta `/app` se encuentran las carpetas `/views`, `/controllers` y `/models` respectivamente, forzando a seguir el patrón obligatoriamente, al separarlo en distintos directorios y diferenciar así la lógica de negocio y la lógica de presentación.

El ciclo de la aplicación comienza en la carpeta `/public/index.php`, que fue redireccionado por `.htaccess` y Apache. Aquí comienza el manejo de los request y el retorno de un response al cliente. Luego `/bootstrap/start.php` instancia una aplicación Laravel, detecta el entorno (environment) y lo inyecta por IoC. El framework carga los Service Provider ( nosotros creamos uno para usar DomPDF), se cargan las rutas y se envía el request a la aplicación, la cual retorna un response al cliente.

Como un resumen de esta sección, la siguiente imagen presenta el recorrido de un request de una aplicación Laravel típica.

## Proyecto de Software

2. Route to appropriate Laravel Controller



## Referencias

Documentación Oficial del Framework CodeIgniter: <http://www.codeigniter.com/userguide3/>

Página Oficial del Framework Symfony: <http://symfony.com>

Página Oficial del Framework Zend: <http://framework.zend.com/>

Documentación oficial Laravel: <http://laravel.com/docs/4.2/>

Cheat Sheet de Laravel : <http://cheats.jesse-obrien.ca/>

Libro de Laravel, Arquitectura MVC de Laravel, <http://laravelbook.com/laravel-architecture/>

Página Oficial del Framework Yii: <http://www.yiiframework.com/>