

Creación de un API RESTful con Slim Framework

Qué es REST y RESTful

Representational State Transfer (REST)

Según la definición puramente teórica, REST es un estilo de arquitectura que abstrae los elementos de dicha arquitectura dentro de un sistema hipermedia distribuido.

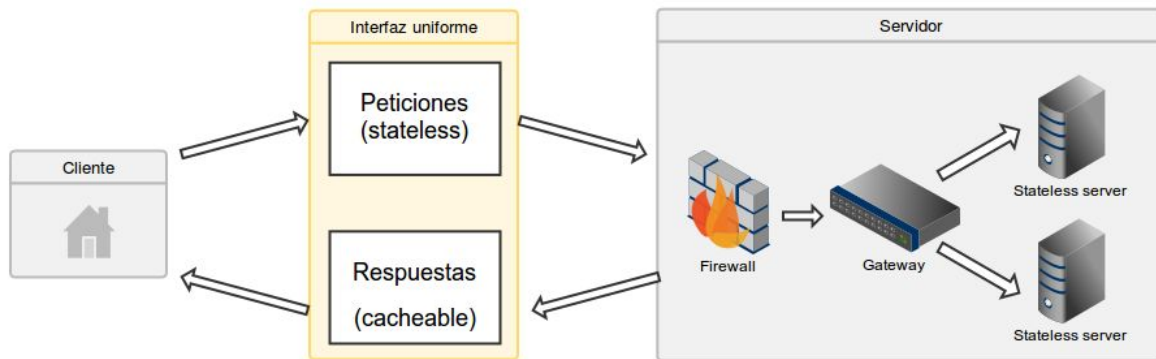
(https://es.wikipedia.org/wiki/Transferencia_de_Estado_Representacional), Pero esto de por sí no nos dice demasiado a no ser que nos dediquemos al estudio teórico de este tipo de cosas. Así que, de nuevo, **¿qué es REST?** Utilizando palabras llanas, podríamos decir que REST es un conjunto de principios, o maneras de hacer las cosas, que define la interacción entre distintos componentes, es decir, las reglas que dichos componentes tienen que seguir. El protocolo más usado que cumple esta definición, es el protocolo HTTP.

Esto quiere decir, por extensión, que toda aplicación web bajo el protocolo HTTP es a su vez una aplicación REST. Sin embargo, como veremos más abajo, eso no implica en absoluto que todas las aplicaciones web sean servicios web RESTful, ya que estas tienen que cumplir una serie de requisitos para ser consideradas tales. Existen, además, otros tipos de métodos para implementar servicios web, que seguramente te suenen, como **RPC**, **SOAP** o **WSDL**. Sin embargo, el uso de dichos mecanismos no se suele recomendar en favor de RESTful, ya que RESTful es mucho más fácil de entender e implementar. No obstante, como siempre, la decisión sobre implementar una tecnología u otra depende en gran medida de las **características del proyecto** en el que estemos implicados, por lo que es recomendable siempre hacer un análisis concienzudo del proyecto y las tecnologías disponibles para decantarnos por una o por otra.

Reglas de la arquitectura REST

REST define una serie de reglas que toda aplicación que pretenda llamarse REST debe cumplir. Como veremos, estas reglas se nos dan ya dadas si vamos a usar el protocolo HTTP, en cualquiera de sus implementaciones (por ejemplo, Apache). No obstante, conviene repasar estos conceptos para tener una visión más profunda, y veremos más adelante que nos ayudará bastante a la hora de implementar nuestros programas:

1. **Arquitectura cliente-servidor:** consiste en una separación clara y concisa entre los 2 agentes básicos en un intercambio de información: el cliente y el servidor. Estos 2 agentes deben ser independientes entre sí, lo que permite una flexibilidad muy alta en todos los sentidos.
2. **Stateless:** esto significa que nuestro servidor no tiene porqué almacenar datos del cliente para mantener un estado del mismo. Esta limitación es sujeto de mucho debate en la industria, incluso ya empiezan a usarse tecnologías relacionadas que implementan el estado dentro de la arquitectura, como WebSockets. Como sabemos, HTTP también cumple esta norma, por lo que estamos acostumbrados ya a hacer uso de protocolos stateless.
3. **Cacheable:** esta norma implica que el servidor que sirve las peticiones del cliente debe definir algún modo de cachear dichas peticiones, para aumentar el rendimiento, escalabilidad, etc. Una vez más, HTTP implementa esto con la cabecera "Cache-control", que dispone de varios parámetros para controlar la cacheabilidad de las respuestas.
4. **Sistema por capas:** nuestro sistema no debe forzar al cliente a saber por qué capas se tramita la información, lo que permite que el cliente conserve su independencia con respecto a dichas capas.
5. **Interfaz uniforme:** esta regla simplifica el protocolo y aumenta la escalabilidad y rendimiento del sistema. No queremos que la interfaz de comunicación entre un cliente y el servidor dependa del servidor al que estamos haciendo las peticiones, ni mucho menos del cliente, por lo que esta regla nos garantiza que no importa quién haga las peticiones ni quien las reciba, siempre y cuando ambos cumplan una interfaz definida de antemano.



REST y HTTP

Vista esta pequeña introducción a los conceptos y reglas básicas de REST, podríamos implementar sin problemas un protocolo que hiciera uso de dicha arquitectura. A pesar de que podría ser un ejercicio más que interesante si lo que buscamos es aprender, no necesitamos hacerlo ya que afortunadamente contamos con uno que implementa REST y que además es la base de Internet. Este protocolo es HTTP. Empezaremos diciendo que para que una aplicación sea REST al 100%, tendrá que implementar 4 principios básicos, y pondremos esto en relación a cómo HTTP implementa dichos principios:

1. **Identificación de recursos:** toda aplicación REST debe poder identificar sus recursos de manera uniforme. HTTP implementa esto usando las llamadas URIs (Uniform resource identifier). Esta es la URL que usamos tradicionalmente, y aunque hay una diferencia sutil entre URLs y URIs (<https://www.yoseomarketing.com/blog/que-es-uri-que-significa-diferencia-url/>), diremos que toda URL es una URI a su vez. Esta identificación del recurso no incluye la representación del mismo, cosa que veremos a continuación.
2. **Recursos y representaciones:** visto que todo recurso debe tener una identificación (URI), REST define también la manera en que podemos interactuar con la representación del mismo, ya sea para editarlo o borrarlo, directamente del servidor. Estas representaciones se dejan a instancias de la implementación final del programa, pero HTTP define distintas cabeceras de tipos, y un contenido en la respuesta, por lo que nuestras aplicaciones pueden enviar el contenido en el formato que quieran, siempre y cuando este contenido contenga la información necesaria para poder operar con el objeto en el caso de que tengamos permiso para hacerlo.

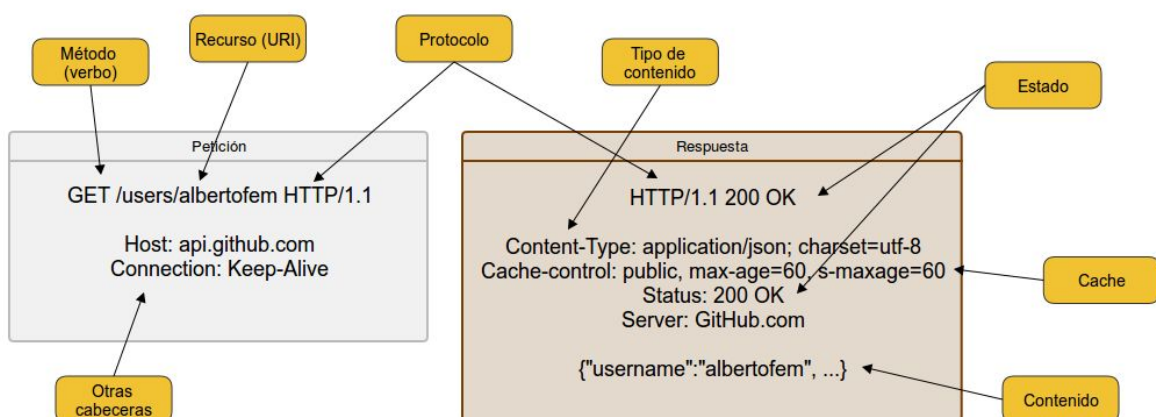
3. **Mensajes autodescriptivos:** cuando hacemos peticiones a un servidor, éste debería devolver una respuesta que nos permita entender sin lugar a duda cual ha sido el resultado de la operación, así como si dicha operación es cacheable, si ha habido algún error, etc. HTTP implementa esto a través del estado y una serie de cabeceras. El cómo se usan estos estados y cabeceras depende por entero de la implementación de nuestro programa, en otras palabras, REST no fuerza el contenido de dichos elementos, por lo que el programa que se ejecuta en el servidor, y que en última instancia accede a los recursos y opera con ellos, tiene la responsabilidad de devolver estados y cabeceras que se correspondan con el estado real de la operación realizada. Esto es importante tenerlo en cuenta, ya que, desgraciadamente, un gran número de aplicaciones y servicios web no respetan esta regla (por lo tanto no pueden ser considerados REST), lo que nos lleva a tener que realizar todo tipo de *workarounds* y cosas por el estilo. En la segunda parte de esta serie veremos un caso práctico de un servicio web que no respeta esta norma, y daremos varias soluciones posibles.
4. **HATEOAS:** por último, y algo que la mayoría de servicios web no cumplen, es la necesidad de incluir en las respuestas del servidor toda aquella información que necesita el cliente para seguir operando con este servicio web. En otras palabras, el cliente *no* tiene porqué saber que cuando obtenemos, por ejemplo, un objeto cualquiera, tenemos además las opciones de modificarlo, o eliminarlo. El servidor debe enlazar a estas operaciones en la respuesta a dicha petición. De esta manera, lo único que necesita saber un cliente sobre una aplicación REST, es el punto de entrada (*endpoint*). Además nos garantiza más independencia entre el cliente y el servidor. Desgraciadamente, este último principio no se implementa en la mayoría de APIs que usamos hoy en día, por lo que, siendo estrictos, podríamos decir que la mayoría de servicios web no son 100% RESTful. No obstante, esto es una limitación menor que en *prácticamente* ningún caso supone un problema.

Servicios web RESTful

Así pues, teniendo claros estos conceptos y cómo se relacionan con el protocolo HTTP, y sabiendo que un servicio web RESTful hace referencia a un servicio web que implementa la arquitectura REST, podemos ya dar una definición concisa, lo cual nos dejará claro cómo tenemos que implementarlo en nuestras aplicaciones. Un servicio web RESTful contiene lo siguiente:

- **URI del recurso.** Por ejemplo: *http://api.servicio.com/recursos/casas/1* (esto nos daría acceso al recurso “Casa” con el ID “1”)
- **El tipo de la representación de dicho recurso.** Por ejemplo, podemos devolver en nuestra cabecera “Content-type: application/json”, por lo que el cliente sabrá que el contenido de la respuesta es una cadena en formato JSON, y podrá procesarla como prefiera. El tipo es arbitrario, siendo los más comunes **JSON**, **XML** y **TXT**.
- **Operaciones soportadas:** HTTP define varios tipos de operaciones (verbos) (https://es.wikipedia.org/wiki/Protocolo_de_transferencia_de_hipertexto), que pueden ser **GET**, **PUT**, **POST**, **DELETE**, **PURGE**, entre otros. Es importante saber para qué están pensados cada verbo, de modo que sean utilizados correctamente por los clientes.
- **Hipervínculos:** por último, nuestra respuesta puede incluir hipervínculos hacia otras acciones que podamos realizar sobre los recursos. Normalmente se incluyen en el mismo contenido de la respuesta, así si por ejemplo, nuestra respuesta es un objeto en JSON, podemos añadir una propiedad más con los hipervínculos a las acciones que admite el objeto.

En la imagen se muestra un ejemplo de una petición al servicio web de GitHub. No se muestra, pero en el contenido de la respuesta *no* se incluían hipervínculos a otras acciones, por lo tanto no es una servicio web RESTful completo.



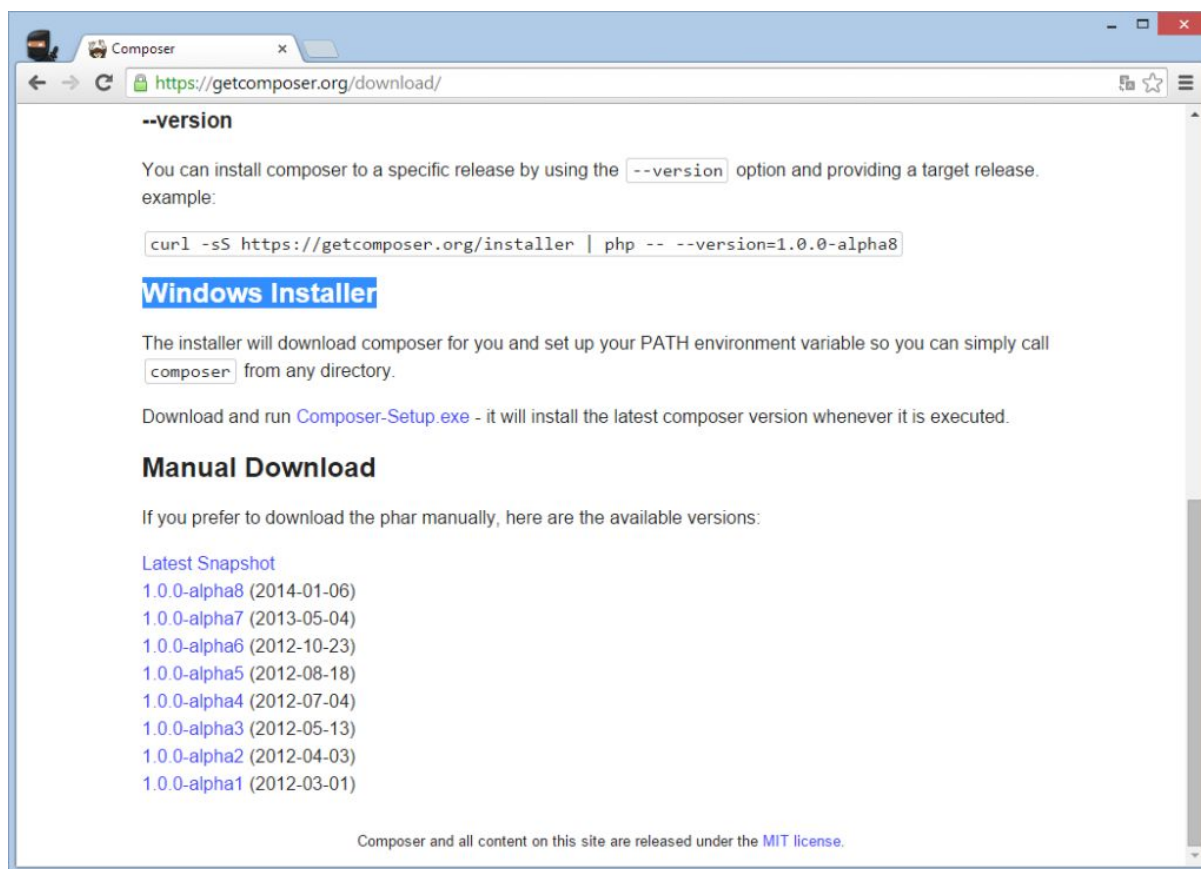
sacado de <http://www.adwe.es/general/colaboraciones/servicios-web-restful-con-http-parte-i-introduccion-y-bases-teoricas#ref>

Como instalar Composer

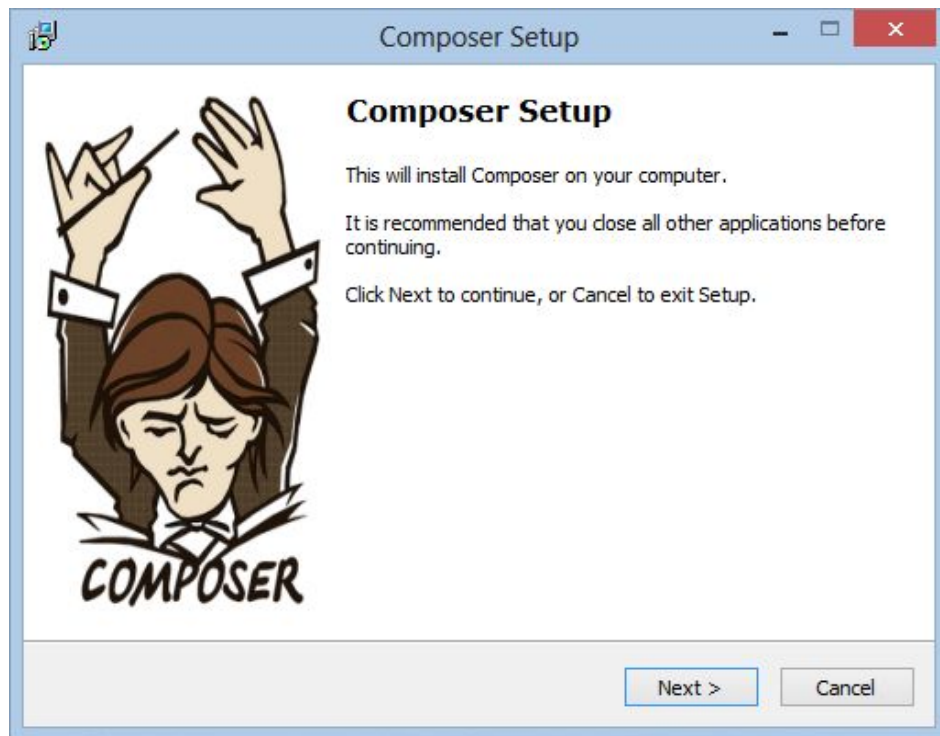
Windows

Composer es una herramienta para gestionar las dependencias en PHP. Te permite declarar las librerías de las cuales tu proyecto depende o necesita y las instala en el proyecto por ti, si deseas saber más acerca de Composer lee el siguiente post: <https://styde.net/que-es-composer-y-como-usarlo>

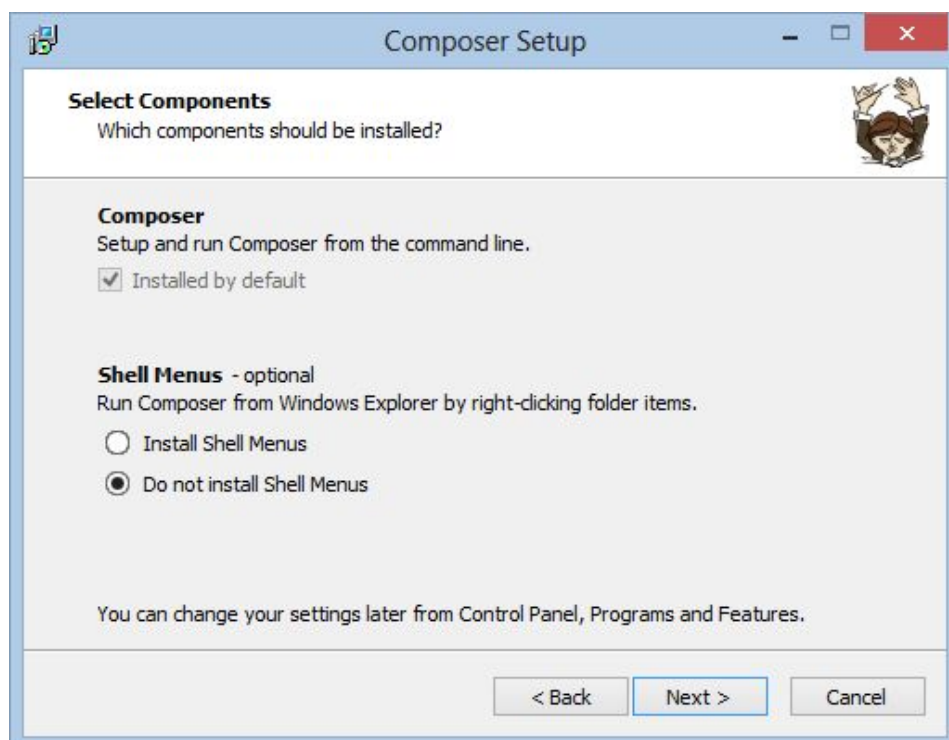
Para instalar Composer en Windows debemos descargarlo de su pagina oficial <https://getcomposer.org/download> y en la sección Windows Installer, haz click en Composer-Setup.exe.



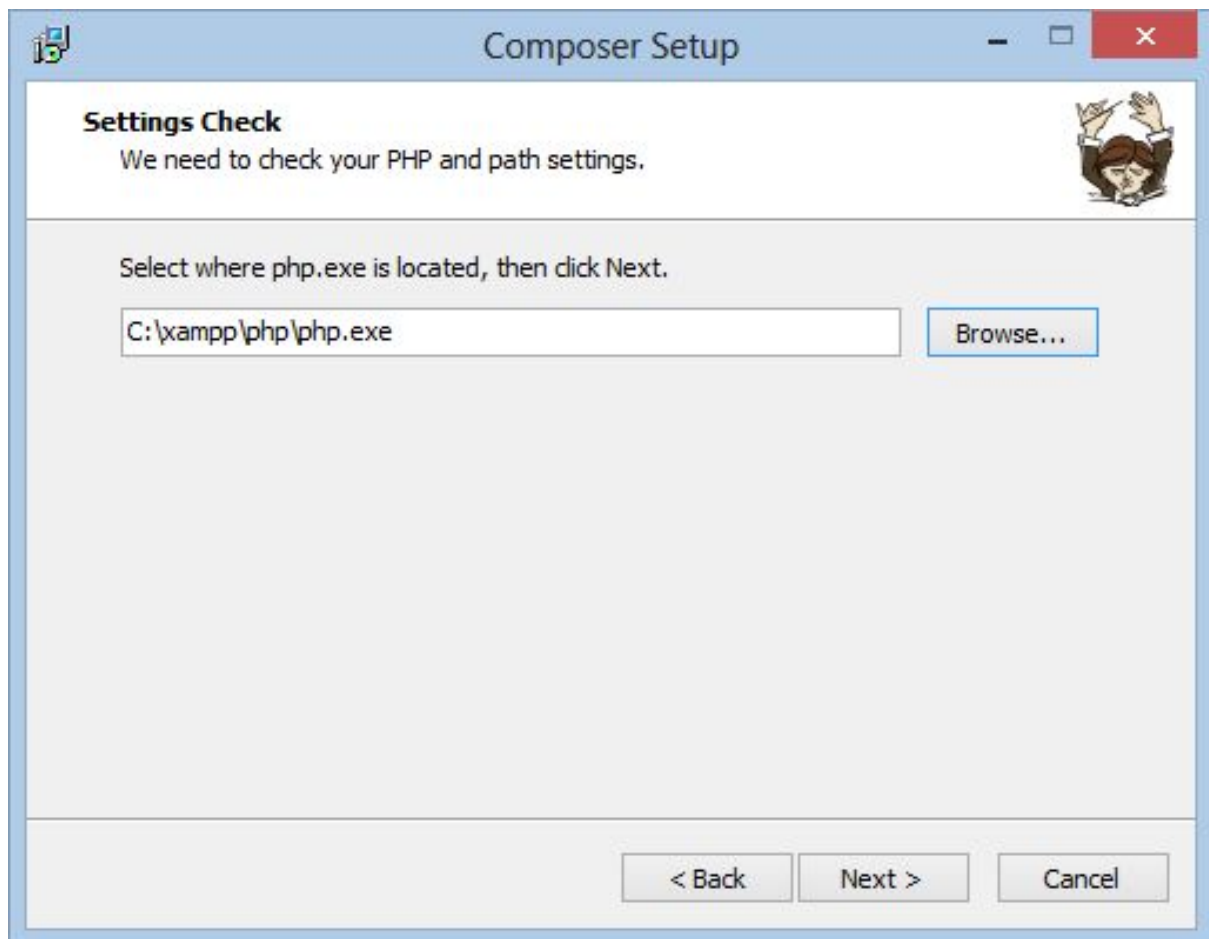
Una vez que la descarga finalice, ejecuta el instalador y haz click en Next.



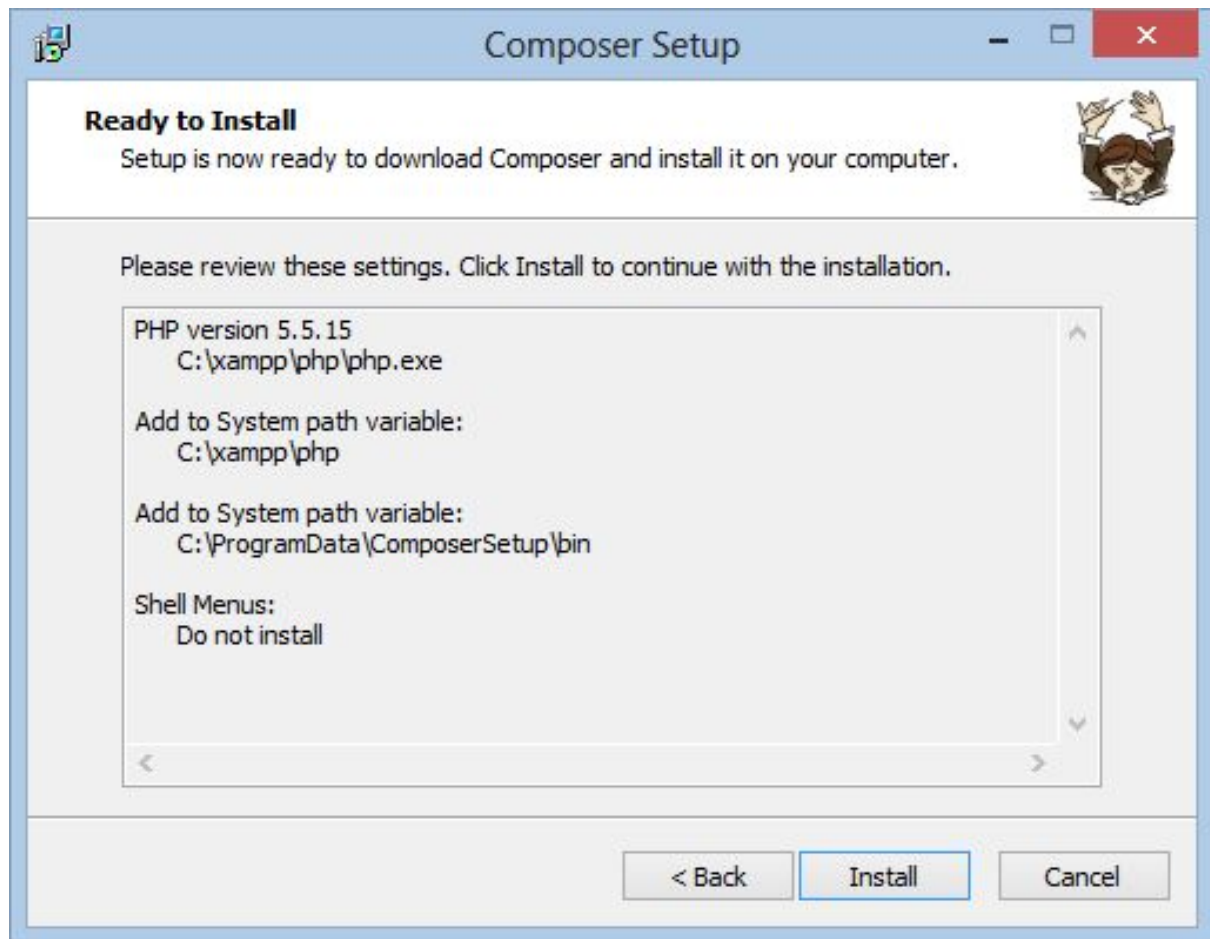
Si quieres administrar tus proyectos mediante el Explorador de Windows puedes seleccionar la opción "Install Shell Menus" aunque lo recomendable es la usar la línea de comandos.



A continuación nos pide que indiquemos la ruta del ejecutable de PHP, en mi caso como estoy trabajando con **XAMPP** (ver página oficial de descarga en la bibliografía) el ejecutable de PHP se encuentra en la ruta C:\xampp\php\ (si usas WAMPP la ruta es C:\wamp\bin\php\php5.5.12) y seleccionas php.exe, luego click en Next.



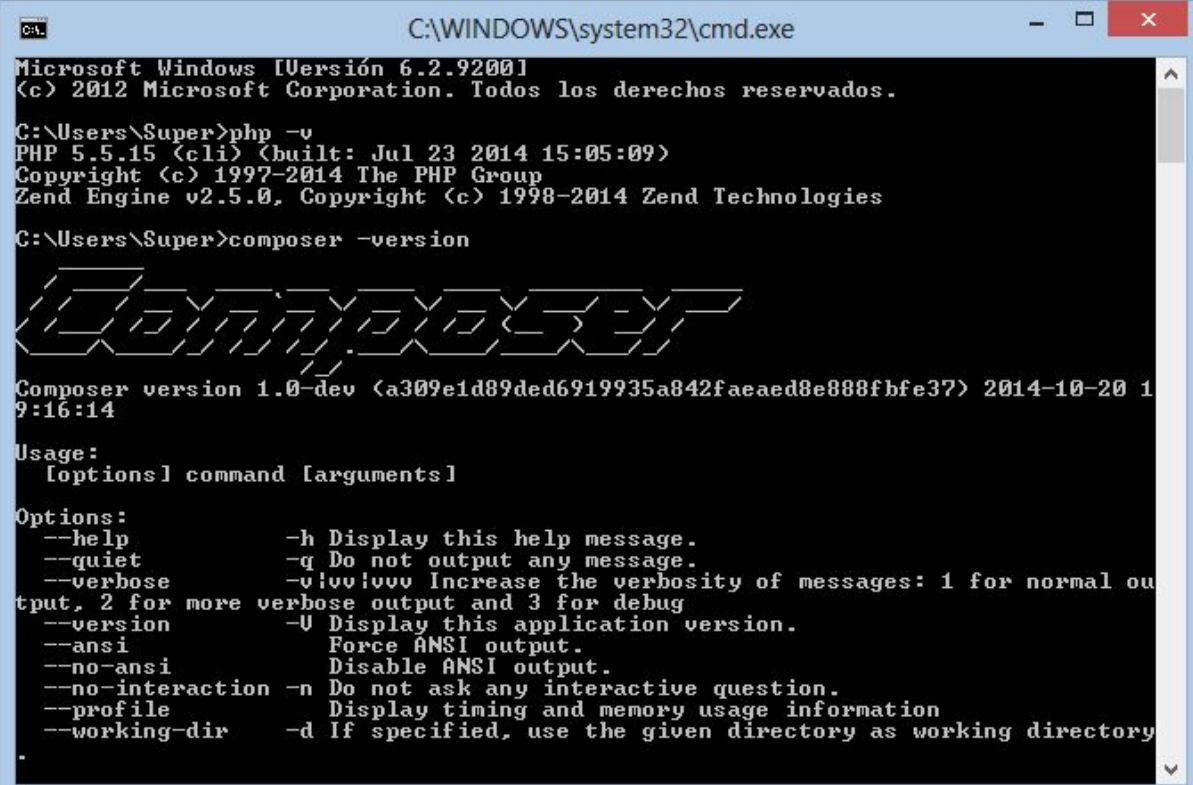
En este punto el instalador de Composer nos muestra la configuración de la instalación, simplemente le damos click a Install.



Una vez esté todo instalado, aparecerán otras donde simplemente debes hacer click en Next, y posteriormente en Finalizar; después de tantos Next, Next típicos de Windows el instalador de Composer habrá puesto en nuestro PATH global la ruta de la carpeta PHP y su propia carpeta Composer. Esto nos permite trabajar desde consola escribiendo sólo php o composer sin necesidad de indicar la ruta del ejecutable. Para ver que todo está en orden vamos a realizar dos pequeñas pruebas, así que es momento de abrir la consola, y teclear:

1. php -v (tecla Enter)
2. composer -version (tecla Enter)

Esto debería devolver la versión de cada uno, como se ve en la siguiente imagen:



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Versión 6.2.9200]
(c) 2012 Microsoft Corporation. Todos los derechos reservados.

C:\Users\Super>php -v
PHP 5.5.15 (cli) (built: Jul 23 2014 15:05:09)
Copyright (c) 1997-2014 The PHP Group
Zend Engine v2.5.0, Copyright (c) 1998-2014 Zend Technologies

C:\Users\Super>composer -version

Composer version 1.0-dev (a309e1d89ded6919935a842faeaed8e888fbfe37) 2014-10-20 19:16:14

Usage:
  [options] command [arguments]

Options:
  --help           -h Display this help message.
  --quiet          -q Do not output any message.
  --verbose        -v|vv|vvv Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug
  --version        -V Display this application version.
  --ansi           Force ANSI output.
  --no-ansi        Disable ANSI output.
  --no-interaction -n Do not ask any interactive question.
  --profile        Display timing and memory usage information
  --working-dir    -d If specified, use the given directory as working directory
.
```

Con esto ya tenemos Composer instalado y funcionando en Windows, ahora solo nos queda instalar Laravel, veamos cómo hacer esto posible.

Sacado de <https://styde.net/instalacion-de-composer-y-laravel-en-windows/>

macOS

Para instalar en macOS ejecutamos los siguientes comandos:

1. `cd /tmp`
2. `curl -sS https://getcomposer.org/installer | php`
3. `mv composer.phar /usr/local/bin/composer`

Para finalizar, entonces verificamos la versión de composer con el siguiente comando:

1. `composer -v`

```

Last login: Mon Mar  5 22:25:56 on ttys000
MacBook-Pro-de-Yhoan:~ yhoanandresgaleanourrea$ composer -v

Composer version 1.6.3 2018-01-31 16:28:17

Usage:
  command [options] [arguments]

Options:
  -h, --help                Display this help message
  -q, --quiet               Do not output any message
  -V, --version             Display this application version
  --ansi                   Force ANSI output
  --no-ansi                Disable ANSI output
  -n, --no-interaction     Do not ask any interactive question
  --profile                Display timing and memory usage information
  --no-plugins             Whether to disable plugins.
  -d, --working-dir=WORKING-DIR If specified, use the given directory as working directory.
  -v|vv|vvv, --verbose    Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug

Available commands:
  about      Shows the short information about Composer.
  archive    Creates an archive of this composer package.
  browse     Opens the package's repository URL or homepage in your browser.
  check-platform-reqs Check that platform requirements are satisfied.
  clear-cache Clears composer's internal package cache.
  clearcache Clears composer's internal package cache.
  config     Sets config options.
  create-project Creates new project from a package into given directory.
  depends    Shows which packages cause the given package to be installed.
  diagnose   Diagnoses the system to identify common errors.
  dump-autoload Dumps the autoloader.
  dumpautoload Dumps the autoloader.
  exec       Executes a vendored binary/script.
  global     Allows running commands in the global composer dir ($COMPOSER_HOME).
  help       Displays help for a command
  home       Opens the package's repository URL or homepage in your browser.
  info       Shows information about packages.
  init       Creates a basic composer.json file in current directory.
  install    Installs the project dependencies from the composer.lock file if present, or falls back on the composer.json.
  licenses   Shows information about licenses of dependencies.
  list       Lists commands
  outdated   Shows a list of installed packages that have updates available, including their latest version.
  prohibits  Shows which packages prevent the given package from being installed.
  remove     Removes a package from the require or require-dev.
  require    Adds required packages to your composer.json and installs them.
  run-script Runs the scripts defined in composer.json.
  search     Searches for packages.

```

Antes de comenzar con la creación de un proyecto con **SLIM** veamos algunos conceptos claves.

Namespaces

En su definición más aceptada, los espacios de nombres (En inglés **namespaces**) son una manera de encapsular elementos. Se pueden ver como un concepto abstracto en muchos aspectos. Por ejemplo, en cualquier sistema operativo, los directorios sirven para agrupar ficheros relacionados, actuando así como espacios de nombres para los ficheros que contienen.

Como ejemplo, el fichero **foo.txt** puede existir en los directorios **/home/greg** y **/home/otro**, pero **no pueden coexistir dos copias de foo.txt** en el mismo directorio. Además, para acceder al fichero **foo.txt** fuera del directorio **/home/greg**, se debe anteponer el nombre del directorio al nombre del fichero, empleando el separador de directorios para así obtener **/home/greg/foo.txt**. Este mismo principio se extiende a los espacios de nombres en el mundo de la programación.

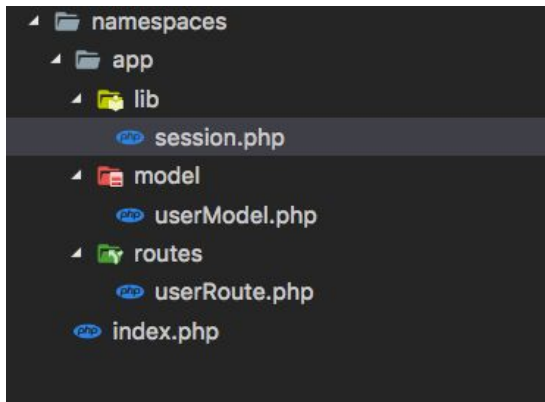
En el mundo de PHP, los espacios de nombres están diseñados para solucionar dos problemas con los que se encuentran los autores de bibliotecas y de aplicaciones al crear elementos de código reusable, tales como clases o funciones:

1. El conflicto de nombres entre el código que se crea y las clases/funciones/constantes internas de PHP o las clases/funciones/constantes de terceros.

2. La capacidad de apodar (o abreviar) Nombres_Extra_Largos diseñada para aliviar el primer problema, mejorando la legibilidad del código fuente.

Los espacios de nombres de PHP proporcionan una manera para agrupar clases, interfaces, funciones y constantes relacionadas. **Veamos un ejemplo:**

Creemos la siguiente estructura:



Para cada uno de los archivos generemos el siguiente contenido:

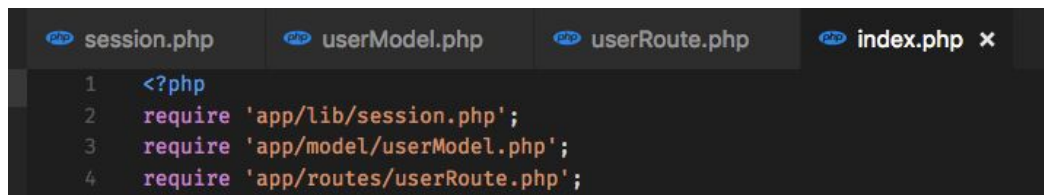
```
session.php x userModel.php userRoute.php
1  <?php
2
3  namespace App\Lib;
4
5  class session
6  {
7
8  }
9
10
11
```

```
session.php userModel.php x userRoute.php
1  <?php
2
3  namespace App\Model;
4
5  class userModel
6  {
7
8  }
```

```
session.php userModel.php userRoute.php x
1  <?php
2
3  namespace App\Routes;
4
5  class userRoute
6  {
7
8  }
```

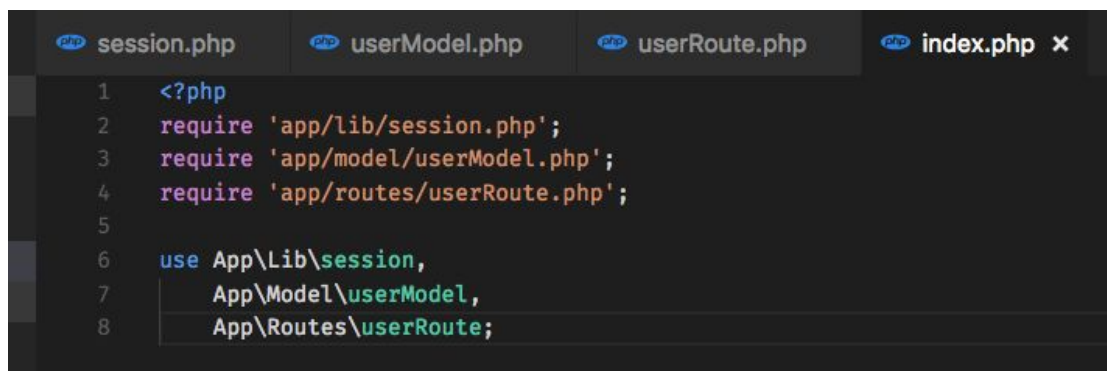
Como podemos notar, se crea una jerarquía para cada una de las clases de nuestro proyecto. Cada una de las clases pertenece a un namespace que agrupa todas las clases que tienen el mismo fin.

Para poder trabajar con estos namespaces, lo primero que debemos hacer es importar los ficheros al archivo en donde queremos trabajar. Para esto escribimos lo siguiente en el archivo index.php que se encuentra en la raíz del proyecto.



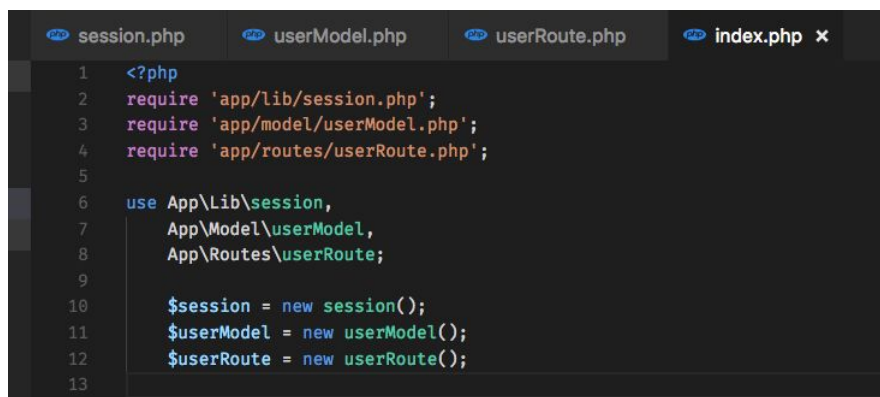
```
1 <?php
2 require 'app/lib/session.php';
3 require 'app/model/userModel.php';
4 require 'app/routes/userRoute.php';
```

Lo normal en php, es que después de importar los ficheros con los que deseamos trabajar, pasemos a crear las instancias de las clases o realizar operaciones con el código importado. En nuestro caso, esto no funcionaria porque debemos especificarle a php cuales de los namespaces de los archivos importados vamos a usar. Para esto entonces utilizamos la palabra reservada **use** y llamamos a los namespaces antes creados.



```
1 <?php
2 require 'app/lib/session.php';
3 require 'app/model/userModel.php';
4 require 'app/routes/userRoute.php';
5
6 use App\Lib\session,
7     App\Model\userModel,
8     App\Routes\userRoute;
```

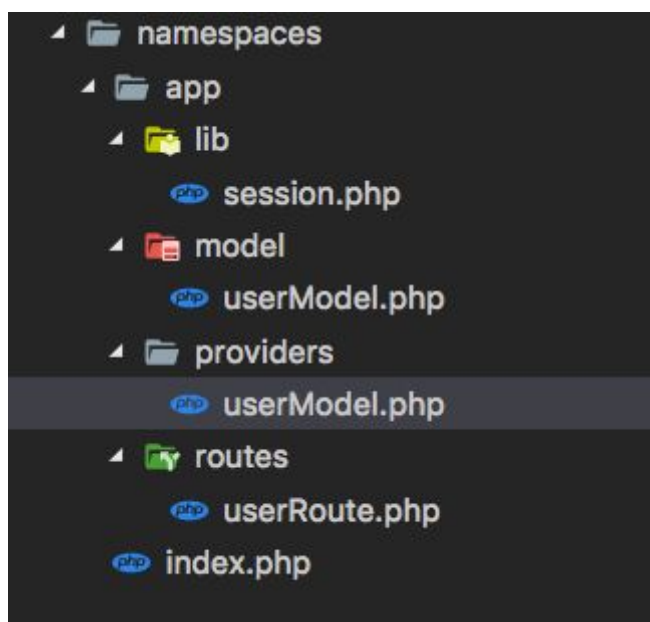
De esta manera, podemos entonces crear instancias de nuestras clases.



```
1 <?php
2 require 'app/lib/session.php';
3 require 'app/model/userModel.php';
4 require 'app/routes/userRoute.php';
5
6 use App\Lib\session,
7     App\Model\userModel,
8     App\Routes\userRoute;
9
10 $session = new session();
11 $userModel = new userModel();
12 $userRoute = new userRoute();
13
```

Como lo dijimos en la conceptualización de los namespaces, estos vienen para solucionar 2 problemas muy comunes que atormenta a los desarrolladores de librerías o aplicaciones basadas en PHP. Que tal si tuviésemos una carpeta llamada **providers** y dentro de esa carpeta tuviésemos otro archivo y otra clase llamada **userModel**, si no utilizáramos namespaces PHP nos sacaría un error diciendo que la clase con el nombre de userModel ya existe dentro de ese contexto. Con las namespaces se puede solucionar este problema ya que se les puede dar alias para que tomen diferentes nombres al usarlos con los espacios de nombre o simplemente hacer el llamado a la clase utilizando todo el namespace. Veamos:

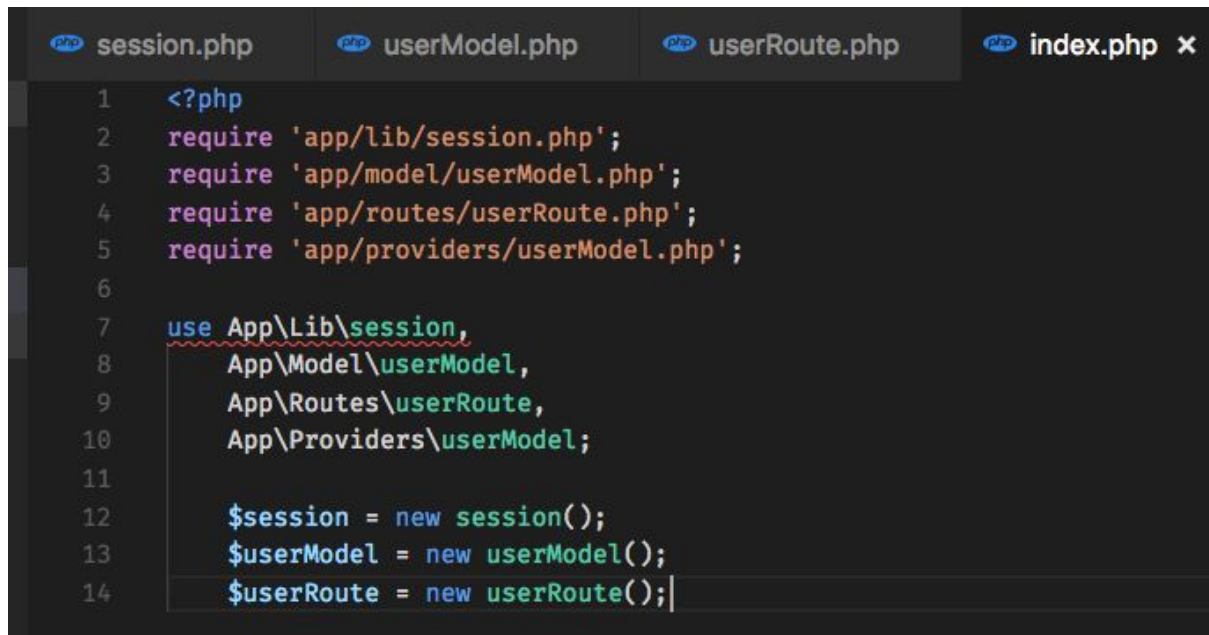
Creemos la carpeta de **providers** y dentro de esta un archivo llamado **userModel.php**



El archivo **userModel.php** tendrá el siguiente contenido:

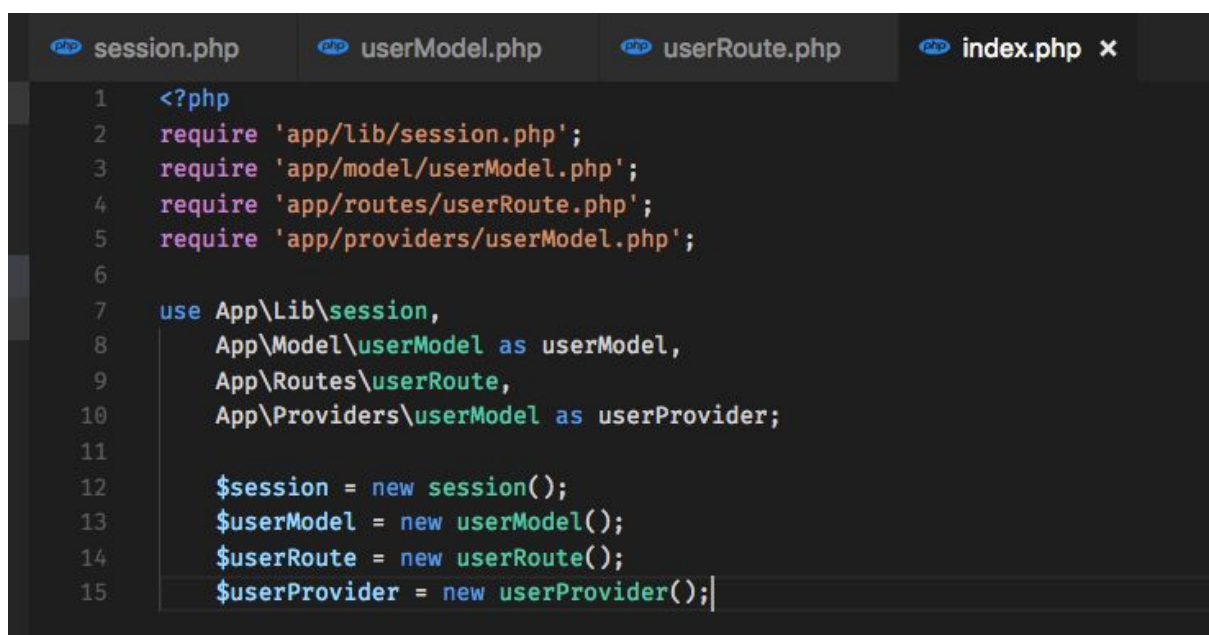
```
1 <?php
2
3 namespace App\Providers;
4
5 class userModel
6 {
7
8 }
```


Ahora, vamos para el archivo `index.php` y hagamos la importación del archivo `userModel.php` de la carpeta `providers` y utilicemos su namespace:



```
1 <?php
2 require 'app/lib/session.php';
3 require 'app/model/userModel.php';
4 require 'app/routes/userRoute.php';
5 require 'app/providers/userModel.php';
6
7 use App\Lib\session,
8     App\Model\userModel,
9     App\Route\userRoute,
10    App\Providers\userModel;
11
12    $session = new session();
13    $userModel = new userModel();
14    $userRoute = new userRoute();|
```

Vemos que PHP automáticamente nos saca un error informando que `userModel` ya está siendo usado por otro namespace. Entonces es aquí donde llegan los alias para ayudarnos a solucionar estos problemas. Veamos:



```
1 <?php
2 require 'app/lib/session.php';
3 require 'app/model/userModel.php';
4 require 'app/routes/userRoute.php';
5 require 'app/providers/userModel.php';
6
7 use App\Lib\session,
8     App\Model\userModel as userModel,
9     App\Route\userRoute,
10    App\Providers\userModel as userProvider;
11
12    $session = new session();
13    $userModel = new userModel();
14    $userRoute = new userRoute();
15    $userProvider = new userProvider();|
```

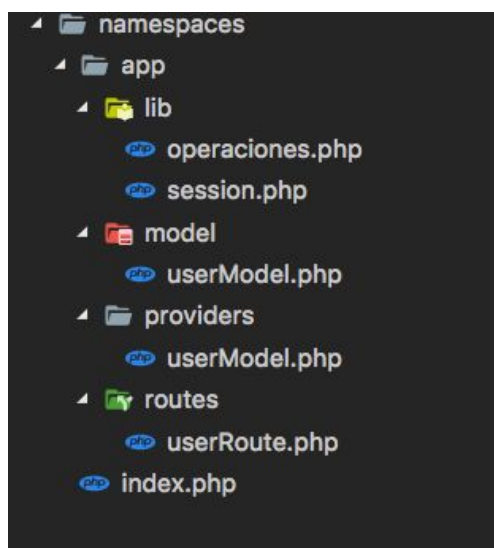
De esta manera nuestras dos clases llamadas `userModel` trabajaran dentro del mismo fichero sin afectar las funcionalidades que cada una de estas precisa.

La otra manera, es no importando los namespaces en el **use** sino que se llaman directamente desde la instancia. Pero ojo, siempre que quieras crear una instancia de esta clase te tocaría estar llamando al namespace:

```
session.php userModel.php userRoute.php index.php x
1  <?php
2  require 'app/lib/session.php';
3  require 'app/model/userModel.php';
4  require 'app/routes/userRoute.php';
5  require 'app/providers/userModel.php';
6
7  use App\Lib\session,
8      App\Model\userModel as userModel,
9      App\Routes\userRoute;
10
11  $session = new session();
12  $userModel = new userModel();
13  $userRoute = new userRoute();
14  $userProvider = new App\Providers\userModel();
15
16
17  $otroUserProvider = new App\Providers\userModel();
18
```

Excepciones y bloque try, catch, finally

Para este ejemplo vamos a crear una clase llamada operaciones en la carpeta de lib:



El contenido de este archivo es:

A screenshot of a code editor with a dark theme. The editor has four tabs at the top: 'userRoute.php', 'index.php', 'operaciones.php' (which is active and has a close button 'x'), and 'userModel.php .../prc'. The code in the active tab is PHP. It starts with a PHP opening tag, followed by a namespace declaration 'App\Lib', and a 'use Exception;' statement. Then, a class named 'operaciones' is defined. Inside the class, there is a static method 'multiplicar' that takes two parameters, '\$num1' and '\$num2'. The method first checks if both parameters are integers using 'is_int'. If not, it throws a new 'Exception' with a custom message. If both are integers, it returns the product of the two numbers. The code is line-numbered from 1 to 19.

```
1 <?php
2
3 namespace App\Lib;
4
5 use Exception;
6
7 class operaciones
8 {
9     public static function multiplicar($num1, $num2){
10
11         if(!is_int($num1) || !is_int($num2)) throw new Exception("Los
12         valores ingresados no son numeros, por favor verifica su tipo.");
13
14         return $num1 * $num2;
15     }
16 }
17
18
19
```

En este caso creamos una clase llamada operaciones, la cual tiene una función static llamada multiplicar (la cual no necesita que la clase sea instanciada para poderse llamar). Esta función recibe 2 parámetros los cuales se esperan que sean números, pero, para mayor seguridad se validan.

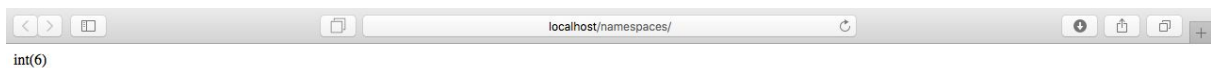
Si por alguna razón se envía un parámetro que no sea número, se lanza una excepción personalizada. Para esto vemos que en la parte superior se hace la instancia del namespace **Exception** de PHP para poder hacer uso del mismo. En el método vemos otra palabra reservada de PHP llamada **throw** la cual para la ejecución del programa para lanzar la correcta excepción.

Si por el contrario, los dos parámetros son numéricos, se retorna la multiplicación de los números.

Ahora vamos a llamar a la función en el archivo index.php para ver lo que sucede al enviarle los parametros correctos:

```
session.php userModel.php userRoute.php index.php x
1 <?php
2 require 'app/lib/session.php';
3 require 'app/model/userModel.php';
4 require 'app/routes/userRoute.php';
5 require 'app/providers/userModel.php';
6 require 'app/lib/operaciones.php';
7
8 use App\Lib\session,
9     App\Model\userModel as userModel,
10    App\Routes\userRoute,
11    App\Providers\userModel as userProvider,
12    App\Lib\operaciones;
13
14 $session = new session();
15 $userModel = new userModel();
16 $userRoute = new userRoute();
17 $userProvider = new userProvider();
18
19 var_dump( operaciones::multiplicar(2, 3) );|
```

Si lo ejecutamos en el navegador, entonces veremos lo siguiente:



Ahora enviamos los parámetros incorrectos:

```
userModel.php userRoute.php index.php x operaciones.php
1 <?php
2 require 'app/lib/session.php';
3 require 'app/model/userModel.php';
4 require 'app/routes/userRoute.php';
5 require 'app/providers/userModel.php';
6 require 'app/lib/operaciones.php';
7
8 use App\Lib\session,
9     App\Model\userModel as userModel,
10    App\Routes\userRoute,
11    App\Providers\userModel as userProvider,
12    App\Lib\operaciones;
13
14 $session = new session();
15 $userModel = new userModel();
16 $userRoute = new userRoute();
17 $userProvider = new userProvider();
18
19 var_dump( operaciones::multiplicar('2', 3) );|
```

Vemos entonces lo que sucede al ejecutarlo:



Esta excepción que saca en el navegador está bien, pues le dijimos que cuando no fueran números realizará dicha operación. Pero esto hace que los usuarios vean el mensaje y además para la ejecución de la aplicación.

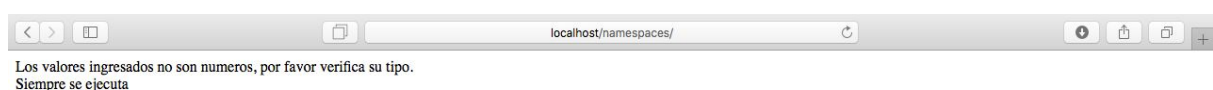
Para esto entonces vamos a controlar ese error. Para esto vamos hacer uso del bloque Try, Catch, Finally. **PHP** tiene un modelo de **excepciones** similar al de otros lenguajes de programación. Una excepción puede ser lanzada ("**thrown**"), y atrapada ("**caught**") dentro de PHP. El código puede estar dentro de un bloque **try** para facilitar la captura de excepciones potenciales. Cada bloque **try** debe tener **al menos un bloque catch o finally** correspondiente. El objeto lanzado debe ser una instancia de la clase **Exception** o una subclase de **Exception**. Intentar lanzar un objeto que no lo sea resultará en un Error Fatal de PHP.

En PHP, se puede utilizar un bloque **finally** después o en lugar de los bloques **catch**. El código de dentro del bloque finally **siempre se ejecutará después de los bloques try y catch**, independientemente de que se haya lanzado una excepción o no, y antes de que la ejecución normal continúe.

Para esto entonces vamos a realizar la captura al error que nos da nuestra función:

```
userModel.php  userRoute.php  index.php x  d
1  <?php
2  require 'app/lib/session.php';
3  require 'app/model/userModel.php';
4  require 'app/routes/userRoute.php';
5  require 'app/providers/userModel.php';
6  require 'app/lib/operaciones.php';
7
8  use App\Lib\session,
9      App\Model\userModel as userModel,
10     App\Routes\userRoute,
11     App\Providers\userModel as userProvider,
12     App\Lib\operaciones;
13
14     $session = new session();
15     $userModel = new userModel();
16     $userRoute = new userRoute();
17     $userProvider = new userProvider();
18
19     try{
20
21         var_dump( operaciones::multiplicar('2', 3) );
22
23     }catch(Exception $e){
24
25         echo $e->getMessage().'\n';
26
27     }finally{
28         echo 'Siempre se ejecuta';
29     }
30
31
```

De esta manera, el código que queremos que se ejecute y estar pendiente de que no ocurran errores es el de la operación multiplicar. El bloque catch está pendiente de los errores y cuando estos ocurran, entonces muestra un mensaje. El bloque finally siempre se ejecuta ya sea que no ocurran o ocurran errores.



FluentPDO

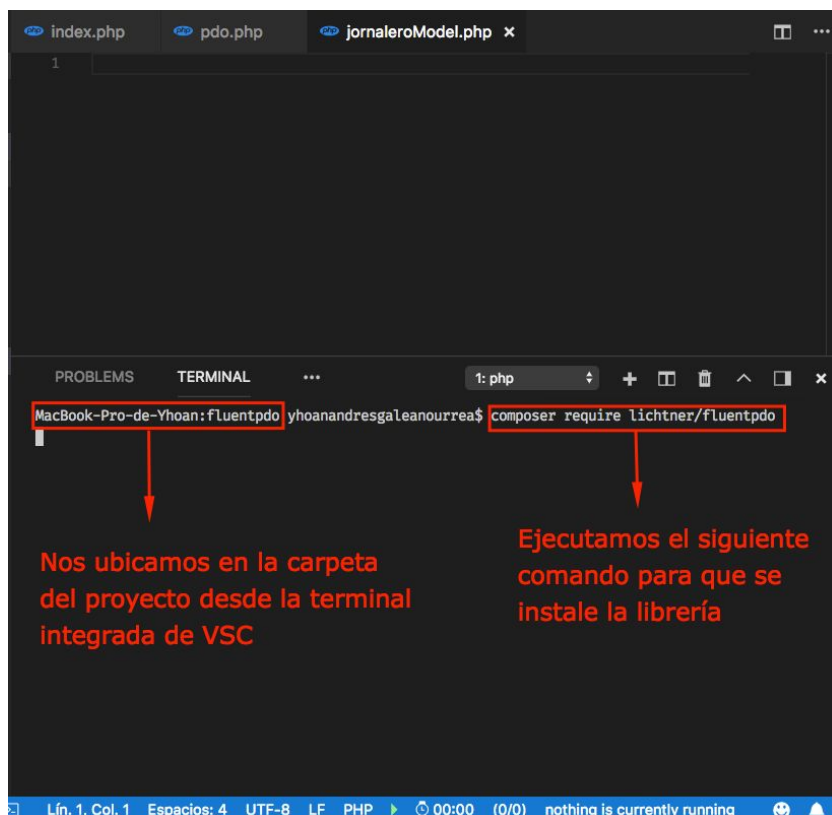
FluentPDO es una librería para PHP y basada en PDO, el driver de PHP para conectarse a base de datos. Esta librería nos permite agilizar y ahorrar código a la hora de realizar las operaciones CRUD a la base de datos. A esta librería se le

conoce como Query Builder el cual nos permite construir las consultas desde PHP y estas son ejecutadas en MySQL.

Para esto vamos a realizar un ejemplo. Creemos la siguiente estructura

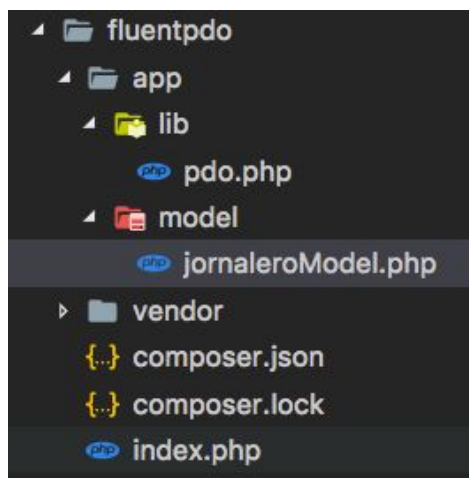


Para instalar FluentPDO lo hacemos via composer, veamos cómo:



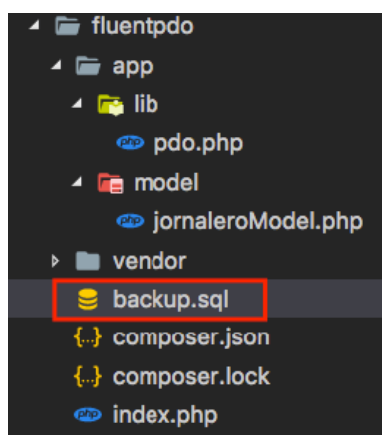
Luego de ejecutar estos comandos veremos algo como lo siguiente en la consola y en nuestro proyecto:

```
PROBLEMS    TERMINAL    ...    1: bash
MacBook-Pro-de-Yhoan:fluentpdo yhoanandresgaleanourrea$ composer require lichtner/fluentpdo
Using version ^1.1 for lichtner/fluentpdo
./composer.json has been created
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
- Installing lichtner/fluentpdo (v1.1.2): Downloading (100%)
Writing lock file
Generating autoload files
MacBook-Pro-de-Yhoan:fluentpdo yhoanandresgaleanourrea$
```



Si vemos composer creo una carpeta llamada vendor y dos archivos llamados composer.json y composer.lock los cuales saben los diferentes paquetes que se han integrado con el proyecto.

Antes de comenzar a escribir el código PHP para utilizar FluentPDO, vamos a realizar el siguiente código en un archivo ubicado en la raíz de nuestra carpeta llamado **backup.sql**:

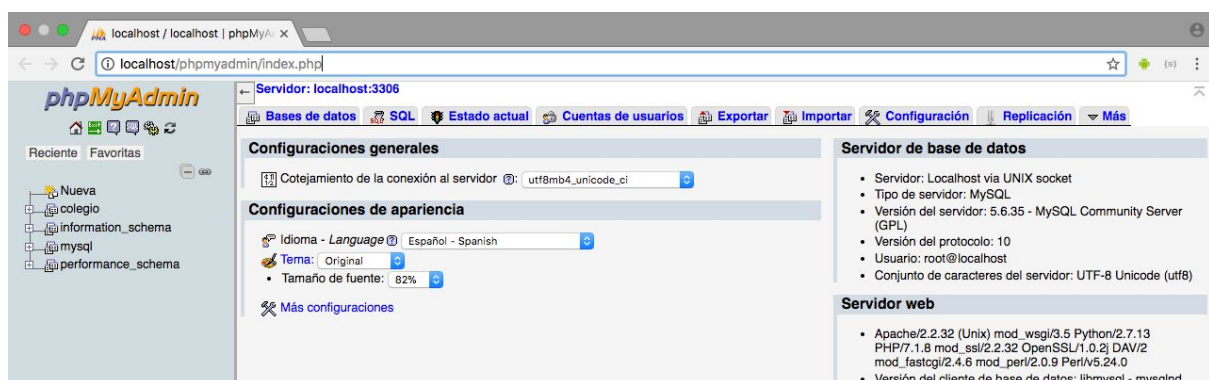



```

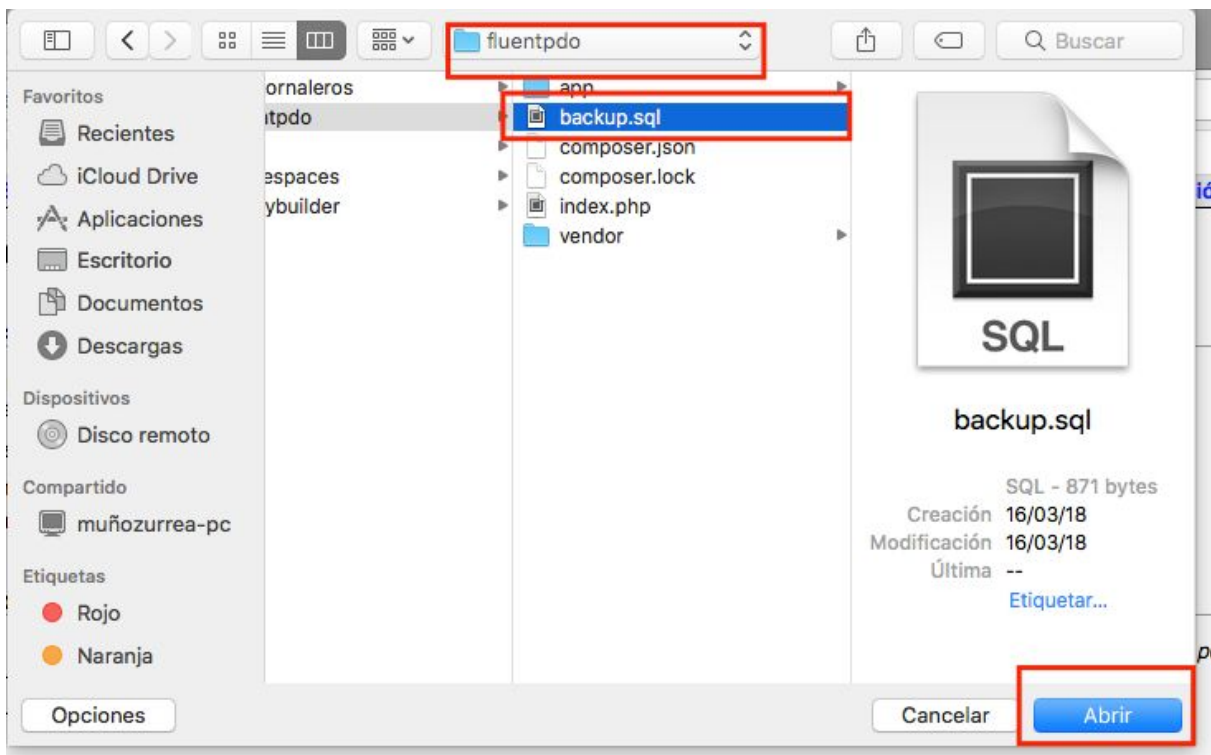
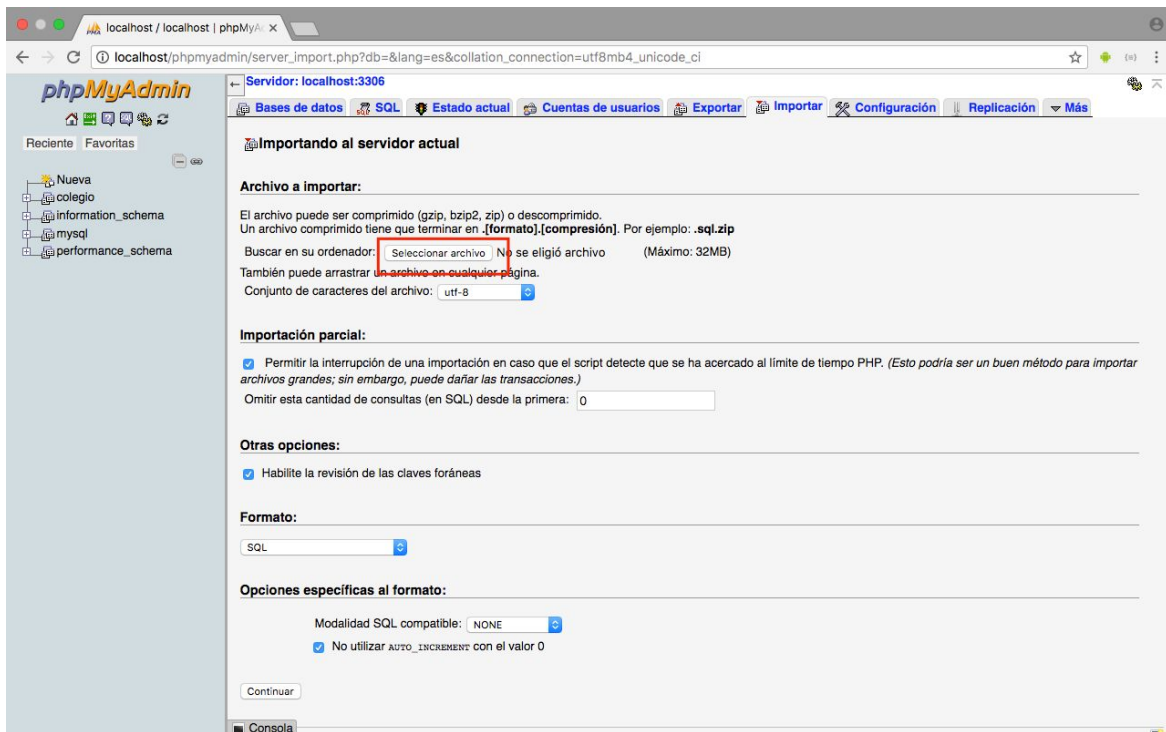
1  -- Volcando estructura de base de datos para jornalero
2  CREATE DATABASE IF NOT EXISTS `jornalero` /*!40100 DEFAULT CHARACTER SET
   utf8 COLLATE utf8_spanish_ci */;
3  USE `jornalero`;
4
5  -- Volcando estructura para tabla jornalero.jornalero
6  CREATE TABLE IF NOT EXISTS `jornalero` (
7    `id` int(11) NOT NULL AUTO_INCREMENT,
8    `nombre` varchar(50) NOT NULL,
9    `correoElectronico` varchar(50) NOT NULL,
10   `fechaNacimiento` date NOT NULL,
11   `peso` int(11) NOT NULL,
12   `creado` datetime DEFAULT CURRENT_TIMESTAMP,
13   `actualizado` datetime DEFAULT NULL,
14   `eliminado` datetime DEFAULT NULL,
15   PRIMARY KEY (`id`)
16 ) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=latin1;
17
18 INSERT INTO `jornalero` (`id`, `nombre`, `correoElectronico`,
19   `fechaNacimiento`, `peso`) VALUES
20   (1, 'Julian Gomez', 'julian@gmail.com', '1994-01-07', 65),
21   (2, 'Sara Lopez', 'sara@gmail.com', '1993-01-28', 78);
22

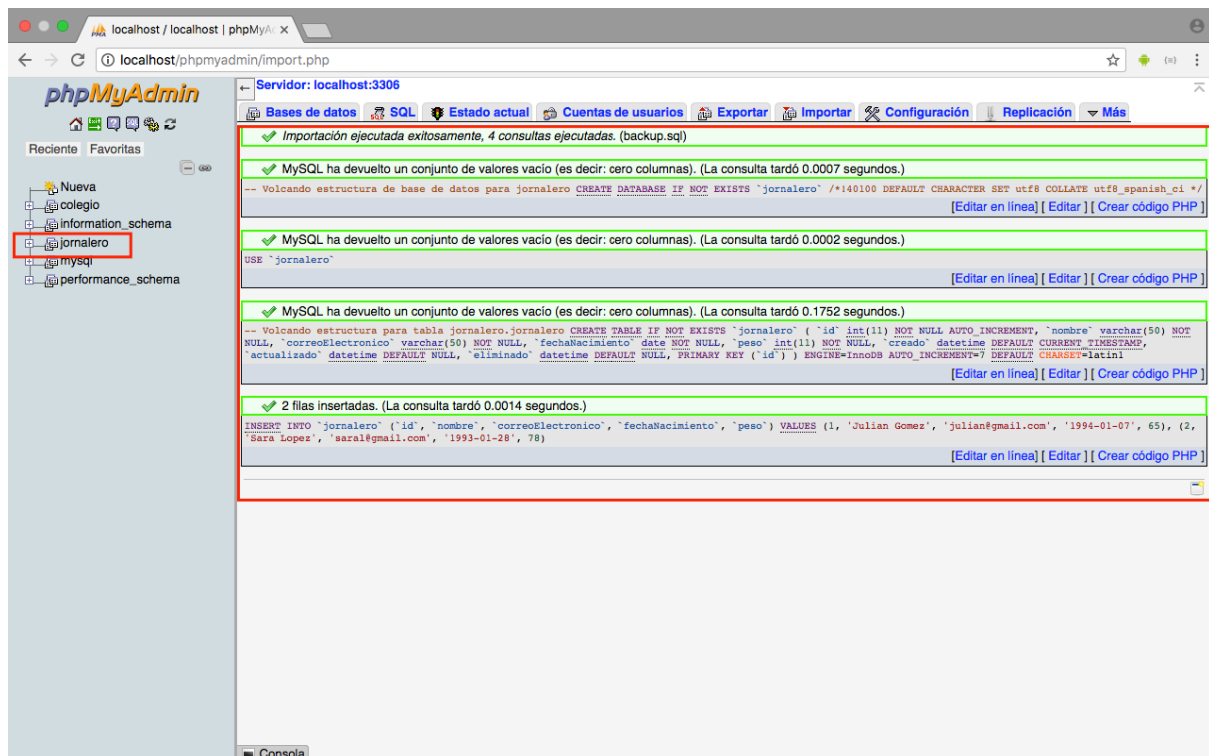
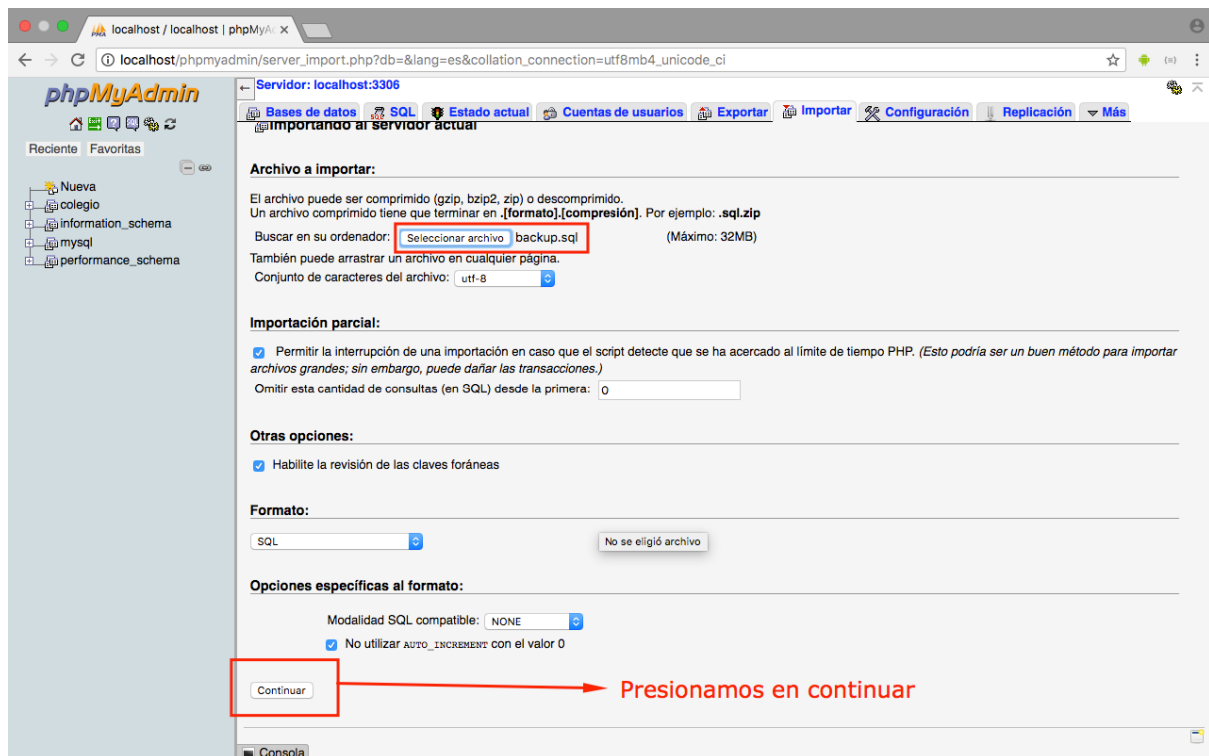
```

Después de esto, vamos a realizar la importación de este archivo a MySQL. Para eso en nuestro navegador vamos a ingresar con la siguiente url:



Este es nuestro administrador de base de datos. El cual nos permite trabajar con el motor de base de datos y realizar las diferentes operaciones. Nos dirigimos a darle click en el menú superior donde dice Importar y luego vamos a seleccionar el archivo llamado backup.sql que escribimos hace poco para que cree la base de datos.





Ahora si, ya teniendo la base de datos importada y FluentPDO instalado, vamos para el archivo **pdo.php** y escribimos lo siguiente:

```
index.php Database.php pdo.php x jornaleroModel.php
1 <?php
2 namespace App\Lib;
3
4 use FluentPDO,
5     PDO;
6
7 class database
8 {
9     public static function getConnection(){
10         try
11         {
12             $pdo = new PDO('mysql:host=localhost;dbname=jornalero;
13                           charset=utf8', 'root', 'root');
14             $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
15             $pdo->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE,
16                               PDO::FETCH_OBJ);
17
18             $connections = array(
19                 'pdo' => $pdo,
20                 'fluentPDO' => new FluentPDO($pdo)
21             );
22
23             return $connections;
24         }
25         catch(Exception $e)
26         {
27             die($e->getMessage());
28         }
29     }
30 }
```

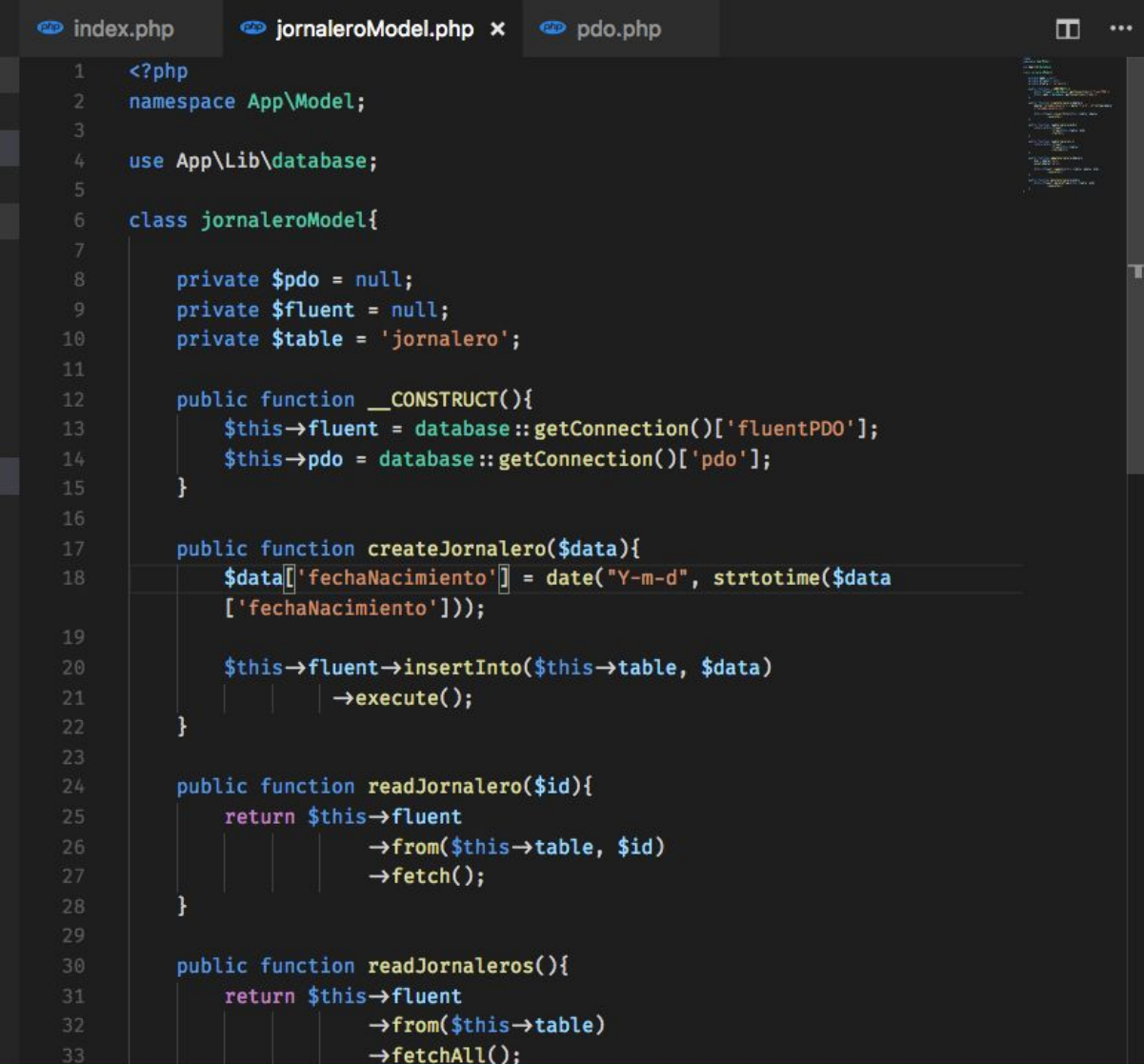
Como vemos en el código anterior, creamos un namespace llamado App\Lib y usamos los namespaces de FluentPDO y PDO. Creamos una clase llamada Database y dentro de esta una función estática llamada getConnection en la cual configuramos el driver de **PDO**. Si vemos todo se encapsula en el bloque **try catch** para estar pendiente de errores que puedan aparecer al intentar hacer la conexión a la base de datos.

Cuando se crea la instancia de PDO vemos que le pasamos un string el cual tiene:

1. La ruta para realizar la conexión a la base de datos.
 - a. En primer lugar le decimos con cual de los motores va a trabajar
 - b. El host en el cual se encuentra la base de datos (en nuestro caso localhost o 127.0.0.1 que es lo mismo)
 - c. Lo siguiente es el nombre de la base de datos a la cual realizaremos las operaciones
 - d. Luego configuramos la encodificación de la devolución y la entrada de datos

- e. Los últimos dos parámetros son del usuario y la contraseña. En mi casa el usuario es root y la contraseña root. En Windows el motor para el usuario por defecto no tiene contraseña, así que en el último parámetro colocas comillas simples sin ningún contenido (").
2. La siguiente línea le especifica a PDO que muestre los errores que ocurran en la base de datos.
3. La siguiente línea le especifica a PDO la forma en que retorna los datos desde la base de datos, en este caso como un objeto.
4. Se crea una variable **\$connections** la cual lleva la instancia de la conexión a PDO y la conexión con FluentPDO a la cual le pasamos la instancia de PDO.

Ya con nuestra conexión vamos a realizar nuestro modelo el cual realizará las operaciones CRUD básicas a la tabla de Jornalero. Para esto necesitamos el siguiente código en el archivo **jornaleroModel.php**



```
1  <?php
2  namespace App\Model;
3
4  use App\Lib\database;
5
6  class jornaleroModel{
7
8      private $pdo = null;
9      private $fluent = null;
10     private $table = 'jornalero';
11
12     public function __CONSTRUCT(){
13         $this->fluent = database::getConnection()['fluentPDO'];
14         $this->pdo = database::getConnection()['pdo'];
15     }
16
17     public function createJornalero($data){
18         $data['fechaNacimiento'] = date("Y-m-d", strtotime($data
19             ['fechaNacimiento']));
20
21         $this->fluent->insertInto($this->table, $data)
22             ->execute();
23     }
24
25     public function readJornalero($id){
26         return $this->fluent
27             ->from($this->table, $id)
28             ->fetch();
29     }
30
31     public function readJornaleros(){
32         return $this->fluent
33             ->from($this->table)
34             ->fetchAll();
35     }
```

```

34     }
35
36     public function updateJornalero($data){
37         $id = $data['id'];
38         unset($data['id']);
39
40         $this->fluent->update($this->table, $data, $id)
41             ->execute();
42     }
43
44     public function deleteJornalero($id){
45         $this->fluent->deleteFrom($this->table, $id)
46             ->execute();
47     }
48 }

```

Para hacer uso de las operaciones CRUD creadas en el modelo, vamos a ir a nuestro archivo **index.php** para hacer la importación de los archivos, usar los namespaces, crear la instancia del modelo y llamar los métodos.

```

index.php x  jornaleroModel.php  pdo.php
1  <?php
2
3  require 'vendor/autoload.php';
4
5  require 'app/lib/pdo.php';
6  require 'app/model/jornaleroModel.php';
7
8  use App\Model\jornaleroModel;
9
10 $jm = new jornaleroModel();
11
12 $jm->createJornalero([
13     'nombre' => 'Juan David Serna',
14     'correoElectronico' => 'jds@gmail.com',
15     'fechaNacimiento' => '1989-02-11',
16     'peso' => 78
17 ]);
18
19 var_dump($jm->readJornalero(2));
20
21 var_dump($jm->readJornaleros());
22
23 $jm->updateJornalero([
24     'id' => 2,
25     'nombre' => 'Sara Valentina Martinez',
26     'correoElectronico' => 'sara@hotmail.com'
27 ]);
28
29 $jm->deleteJornalero(2);|

```

Creación del Proyecto con Slim Skeleton

Para la creación de un proyecto de slim base, ejecutamos el siguiente comando (Tanto en Windows como en macOS). Para esto te debes de posicionar en la carpeta de htdocs en caso de que estén trabajando con XAMPP:

1. `cd Applications/MAMP/htdocs`
2. `composer create-project slim/slim-skeleton api_jornaleros`

O en Windows:

1. `cd C:/xampp/htdocs`
2. `composer create-project slim/slim-skeleton api_jornaleros`



Con el comando anterior, le especificamos a Slim que descargara una dependencia llamada Slim Skeleton, la cual me crea un proyecto con una arquitectura básica de la utilización del Framework para trabajar y no perderse.

- **logs:** mediante la dependencia **Monolog**, podemos hacer trace a nuestras rutas para ir guardando en el log posible errores, o cosas que queramos hacer seguimiento.
- **public:** se encuentra nuestro index.php que inicia toda nuestra API.
- **src:** Es la carpeta de las configuraciones y recomendada por slim skeleton para trabajar, eso sí, en una aplicación pequeña. Para nuestro caso no aplica mucho ya que vamos a realizar una arquitectura que sea escalable y que permita trabajar con proyectos grandes.
- **templates:** es el equivalente a una carpeta views en MVC, pero como vamos a trabajar RESTful no vamos a usar esto. Podríamos crear una ruta que muestre HTML puro, tal vez para tema de documentación posiblemente.
- **vendor:** las dependencias manejadas para nuestro proyecto vía Composer.

Algunos tips:

- Las dependencias me permiten crear instancias globales para usarlas en toda la aplicación. Un ejemplo de esta es **Monolog** o crear la conexión a PDO y dejarla pública para que cualquier otro componente de la API pueda accederla.
- En las configuraciones tenemos una propiedad llamada `displayErrorDetails` la cual me permite visualizar los errores en la aplicación y que cualquier desarrollador con conocimientos del framework puede comprender. Importe que cuando se pase el proyecto para producción esta propiedad se establezca en false.
- En Slim Skeleton se visualiza una template en la raíz, pero ojo, recordemos que no vamos a realizar una página web, vamos a trabajar con servicios. De esta manera podríamos mostrar una página web pero para visualizar la versión del API, los cambios que ha tenido, los datos que retorna u cualquier otra información relacionada con el funcionamiento.
- Como estamos creando nuestra API, es innecesario crear un cliente para consumir y probar cada uno de los recursos, para esto, instalamos POSTMAN un cliente que me permite consumir cada uno de estos recursos, pasando parámetros y obteniendo las respuestas. En la bibliografía está en el enlace a la aplicación.

Routes y configuración

Sencillas


```

22 $app->any('/users/index', function (Request $request, Response $response) {
23
24     $this->logger->info("Jornaleros App - Ruta '/users/'");
25     return json_encode(
26         array(
27             'mensaje' => 'se puede acceder desde cualquier verbo'
28         )
29     );
30 });
31
32 // Routes de Ejemplo
33
34 $app->get('/users/getAll', function (Request $request, Response $response) {
35
36     $this->logger->info("Jornaleros App - Ruta '/users/getAll'");
37     return 'Obteniendo el usuario';
38 });
39
40 $app->get('/users/get/{id}', function (Request $request, Response $response, array
41     $args) {
42
43     $this->logger->info("Jornaleros App - Ruta '/users/get/{id}'");
44     return 'Obteniendo el usuario ' . $args['id'];
45 });
46
47 $app->post('/users/insert', function (Request $request, Response $response) {
48
49     $body = $request->getBody();
50     $parsedBody = $request->getParsedBody();
51
52
53
54

```

En grupos

```

16 });
17
18 $app->group('/users/', function () {
19
20     $this->any('/', function (Request $request, Response $response) {
21
22         $this->logger->info("Jornaleros App - Ruta '/users/'");
23         return json_encode(
24             array(
25                 'mensaje' => 'Raiz - se puede acceder desde cualquier verbo'
26             )
27         );
28     });
29
30     $this->get('getAll', function (Request $request, Response $response) {
31
32         $this->logger->info("Jornaleros App - Ruta '/users/getAll'");
33         return 'Obteniendo el usuario';
34     });
35
36     $this->get('get/{id}', function (Request $request, Response $response, array
37         $args) {
38
39         $this->logger->info("Jornaleros App - Ruta '/users/get/{id}'");
40         return 'Obteniendo el usuario ' . $args['id'];
41     });
42
43     $this->post('insert', function (Request $request, Response $response) {
44
45         $body = $request->getBody();
46         $parsedBody = $request->getParsedBody();
47
48
49
50

```

Recordemos que para enviar parámetros a una ruta existen varias formas:

1. Por la url, utilizando el verbo get. Ejm:
users/getAll?nombre=Yhoan&orderBy=nombre

2. Por la url utilizando el verbo get, put, delete pero que pertenece a la URI.
Ejm: users/get/23, /users[/id] (Parametro Opcional), /news[/year][/{month}] (Parámetros opcionales anidados), news[/params:.*)] (Parámetros opcionales ilimitados)
3. Por el cuerpo en los verbos post, put que no se identifican.
4. Por los encabezados. Normalmente varios de los parámetros se envían por defecto, como el tipo de navegador, el localhost, el puerto. Para nuestro caso, enviaremos un token de autenticación para validar que el usuario pueda consumir los recursos que solicitó.

Bibliografía

JSON Web Token (JWT)	http://self-issued.info/docs/draft-ietf-oauth-json-web-token.html
JWT Claims	https://self-issued.info/docs/draft-ietf-oauth-json-web-token.html#rfc.section.4.1
Explicación de JWT	http://anexsoft.com/p/125/autenticacion-usando-json-web-token
Códigos de Estado Respuestas HTTP	https://es.wikipedia.org/wiki/Anexo:Códigos_de_estado_HTTP
Página Oficial de Slim Framework	https://www.slimframework.com
Documentación de Slim Framework	https://www.slimframework.com/docs/
Página Oficial de XAMPP	https://www.apachefriends.org/es/index.html
Postman (Cliente REST para probar el API)	https://www.getpostman.com
Namespaces	http://php.net/manual/es/language.namespaces.basics.php http://php.net/manual/es/language.namespaces.rationale.php
Captura de Excepciones	http://php.net/manual/es/language.exceptions.php
Documentación de las routes	https://www.slimframework.com/docs

	s/v3/objects/router.html
--	--