Desarrollo web en entorno servidor

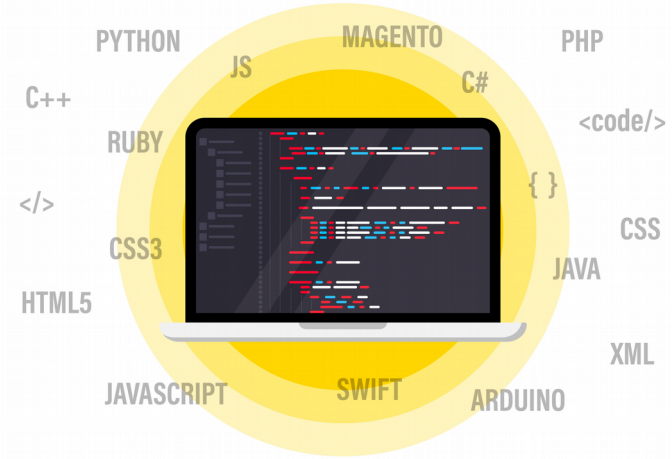
Unidad 10: Desarrollo de aplicaciones web híbridas

ÍNDICE

INTRODUCCIÓN

Como hemos visto en unidades didácticas anteriores, cuando trabajamos de modo profesional en el desarrollo aplicamos el principio de no intentar inventar la rueda y nos ayudamos de frameworks.

Hasta ahora hemos aprendido a crear un CRUD (create, read, uptade, delete), pero evidentemente los **frameworks nos pueden ayudar en muchas más tareas**. En esta unidad didáctica veremos como **Laravel**, nuestra framework de cabecera, nos fuerza a crear código compacto y reutilizable.



Pero no solo esto, con Laravel también será muy sencillo crearnos APIs propias y consumir APIs de terceros. Y, ya puestos a practicar con Laravel, también veremos cómo usarlo para generar **seeders**, es decir, datos aleatorios de muestra para nuestros proyectos.

OBJETIVOS / CAPACIDADES

En esta unidad de aprendizaje, las capacidades que más se van a trabajar son:

- ✔ **Utilizar herramientas y lenguajes específicos, cumpliendo las especificaciones, para desarrollar e integrar componentes software en el entorno del servidor web.**
- ✔ **Evaluar servicios distribuidos ya desarrollados, verificando sus prestaciones y funcionalidad, para integrar servicios distribuidos en una aplicación web.**
- ✔ **Programar y realizar actividades para gestionar el mantenimiento de los recursos informáticos.**

PROYECTO DE LA UNIDAD

Antes de empezar a trabajar el contenido, te presentamos la **actividad** que está relacionada con esta unidad de aprendizaje. Se trata de un **caso práctico** basado en una **situación real** con la que te puedes encontrar en tu puesto de trabajo. Con esta actividad se evaluará la puesta en práctica de los **criterios de evaluación** vinculados al resultado de aprendizaje que se trabaja en esta unidad. Para realizarla deberás hacer lo siguiente: lee el enunciado que te presentamos a continuación, dirígete al área general del módulo profesional, concretamente a la actividad de evaluación que se encuentra dentro de esta unidad, allí encontrarás todos los detalles sobre fecha y forma de entrega, objetivos... A lo largo de la unidad irás adquiriendo los conocimientos necesarios para ir elaborando este proyecto.

Enunciado:

Almacenando datos de empleados.

Queremos montar una API que nos devuelva información sobre características de móviles, concretamente:

- Nombre.
- Marca.
- Tamaño.
- Precio.

Estos datos deben estar en tu base de datos y gracias a la API que crees un usuario podrá consumirlos. Los datos no se introducirán a mano, sino que deberás o buscar una API que tenga estos datos, algo difícil, o bien scrapear una web como puede ser Amazon, PCComponentes, etc.

Finalmente debes montar un XML para sindicación RSS con todos los productos de tu web y ponerlo en la ruta, por ejemplo, `tu_proyecto/feed`.

1. REUTILIZACIÓN DE CÓDIGO E INFORMACIÓN

Hay varios **criterios** que se deben cumplir cuando se escribe un programa. Un programa bien escrito debe ser:

- **Compacto**: no innecesariamente largo.
- **Legible**: fácil a entender y utilizar por otros.
- **Robusto**: no fallar con entradas inesperadas.
- **Reutilizable**: se puede reutilizar parte del programa.

Más allá de todas las herramientas que nos pueden dar los *frameworks*, la responsabilidad de **escribir código limpio recae en el desarrollador**. En las unidades didácticas 2 y 3 ya vimos cómo utilizar bloques de código y el uso de funciones.

➔ Normalmente **un programa está formado por varios métodos** o funciones. Las ventajas de esto son:

- ✔ **Abstracción**: el usuario de un método no necesita conocer el proceso interno, sólo la entrada y salida.
- ✔ **Reutilización**: un método de un programa se puede reutilizar en otro.
- ✔ **Legibilidad**: un programa es más fácil de entender y escribir si está separado en módulos.
- ✔ **Compactibilidad**: no existen varias copias del mismo código.

A menudo hay que realizar una misma operación en varios programas o en distintas partes del mismo programa. Podemos copiar el código varias veces y manipular las entradas para que funcione en otro programa. No obstante, ¿qué

pasa si hay que modificar ese código? Habrá que cambiarlo en todos los lugares donde se encuentra.

*Por esto es mejor tener **una única vez el código y poder llamarlo** desde donde haga falta. También es una buena práctica no utilizar valores absolutos, sino guardarlos en una variable.*

➔ **Imaginemos un caso práctico real**. Queremos hacer un script para calcular la suma de los primeros 5 números enteros y también de los 7 primeros enteros. Podríamos hacer algo así:

```
<?php
      $suma = 0;

      // Calcula la suma de los 5 primeros enteros
      for($i = 1; $i <= 5; $i++) {
        $suma += $i;
      }
      echo "La suma es: " . $suma;

      // Calcula la suma de los 7 primeros enteros
      $suma = 0;
      for($i = 1; $i <= 7; $i++){
        $suma += $i;
      }
      echo "La suma es: " . $suma;
?>
```

Este código es correcto pero muy ineficiente: repetimos estructuras muy similares, no permite escalarlo con facilidad, etc.

Podríamos reutilizar código encapsulando la estructura repetida en una función, por ejemplo:

```php
<?php
    echo "La suma es: " . sumarNumeros(5);
    echo "La suma es: " . sumarNumeros(7);

    function sumarNumeros($num) {
      $suma = 0;
      for($i = 1; $i <= $num; $i++) {
        $suma += $i;
      }
      return $suma;
    }
?>
```

➔ **¿Qué ha mejorado?**

- ✔ La función `sumarNumeros()` nos permite ahorrarnos código duplicado. El código es más compacto y legible.
- ✔ La función `sumarNumeros()` sólo suma números y devuelve un resultado. No se encarga de mostrarlo por pantalla. El código es abstracto, entra un valor y sale otro valor.
- ✔ Si hay que hacer un cambio es mucho más sencillo. El programa es escalable.

2. ARQUITECTURA DE LAS APLICACIONES WEB HÍBRIDAS

*En la **arquitectura cliente-servidor** que hemos ido estudiando hasta ahora, hemos dado por sentado que el cliente consumiría la información recibida por el servidor con un navegador web. Esto es algo correcto y lógico, pero no es la única forma. En la unidad didáctica 8, ya vimos que los servicios web no tienen por qué respondernos con un documento web (**HTML**), sino que nos pueden devolver una respuesta en formato **XML**, **JSON**, etc.*

Esto lo volveremos a ver en esta misma unidad didáctica cuando hablemos de APIs, pero centrándonos en el título de este apartado: **aplicaciones web híbridas ¿a qué nos referimos?** Para contestar a esta pregunta primero debemos reflexionar en **cómo y dónde consumimos las aplicaciones web**. Y lo cierto es que, normalmente, lo haremos desde un ordenador. O quizás desde un móvil, o desde la Smart TV, o desde el coche, etc.

- Una **aplicación web híbrida** es aquella pensada para que se **ejecute correctamente en todos los dispositivos**, independientemente de la marca, formato, forma, sistema operativo, etc. Y cuando hablamos de "ejecución correcta" no solamente nos referimos a que se adapte a la pantalla.

  Pensemos en una aplicación móvil. Por ejemplo, siempre funcionará, independientemente de si hay conexión a Internet. Quizás no cargue datos actualizados, pero la aplicación debería funcionar. Una aplicación web híbrida debe comportarse de la misma forma.

- Las **características** de las aplicaciones web híbridas incluyen:

- ✔ La capacidad de funcionar esté o no conectado el dispositivo.
- ✔ Integración con el sistema de archivos del dispositivo móvil.
- ✔ Integración con servicios basados en web.
- ✔ Un navegador integrado para mejorar el acceso a contenido dinámico en línea.

➔ Las **ventajas** de las aplicaciones web híbridas incluyen:

- ✔ Funcionar en distintas plataformas.
- ✔ Tiempos de compilación más rápidos en comparación con las aplicaciones nativas.
- ✔ Más barato de desarrollar en comparación con la creación de dos o más versiones de una aplicación nativa para plataformas diferentes.
- ✔ Es más fácil lanzar parches y actualizaciones.
- ✔ Puede trabajar en línea y sin conexión.

➔ Sin embargo, también hay **desventajas**:

- ✔ Variaciones debidas al desarrollo inclinado en una plataforma; por ejemplo, si un equipo de desarrollo inclina su trabajo hacia una plataforma, otra plataforma admitida puede carecer de calidad o sufrir errores.
- ✔ La apariencia de una aplicación puede variar de una plataforma a otra.
- ✔ La necesidad de probar la aplicación en una variedad de dispositivos para garantizar un funcionamiento adecuado.
- ✔ La experiencia del usuario (UX) puede caer si la interfaz de usuario (UI) no es similar a los navegadores a los que el usuario está acostumbrado y está lo suficientemente bien diseñada.

3. INSTALACIÓN DE APLICACIONES WEB EN LOS SERVIDORES

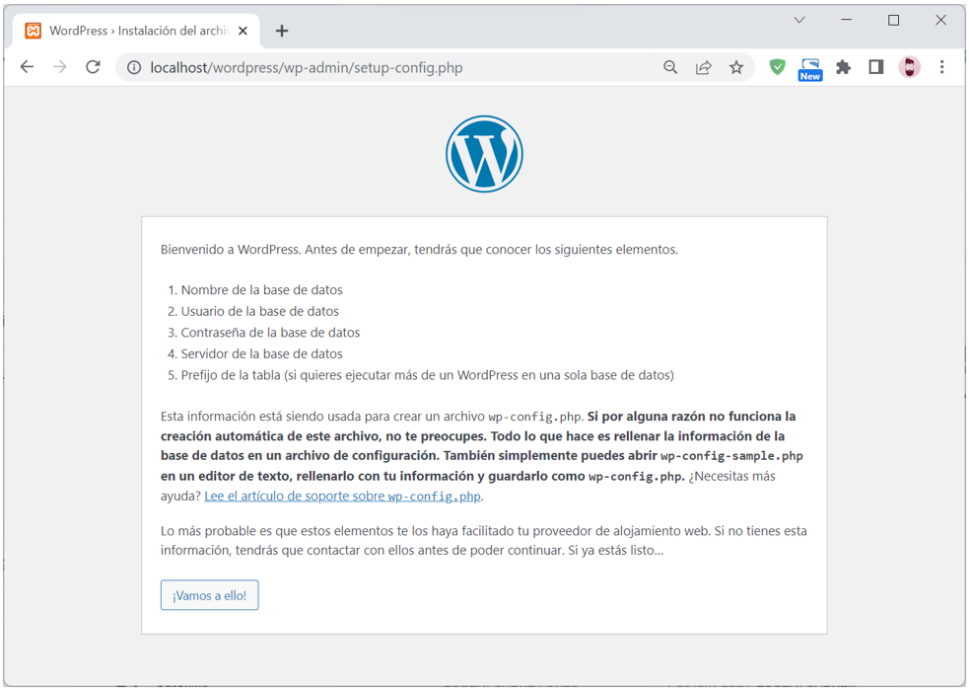
Al principio de curso **creábamos páginas web**: primero estáticas, luego dinámicas. Al ir aprendiendo, **pasamos de las páginas web a las aplicaciones web**. ¿Qué diferencia hay?:

- Una **página web**, puede ser estática o dinámica, y básicamente muestra información al usuario con una interactividad mínima o nula. A veces se las llama landing page, páginas corporativas, etc.
- Por su parte, las **aplicaciones web** son plataformas principalmente interactivas que se centran en que los usuarios realicen acciones. Un ejemplo de una aplicación web es WordPress, quizás el CMS más famoso del momento. Este es el que usaremos para nuestras pruebas, pero aplicaciones web hay infinitas y para distintos propósitos: Drupal, Joomla, Moodle, Odoo, etc.

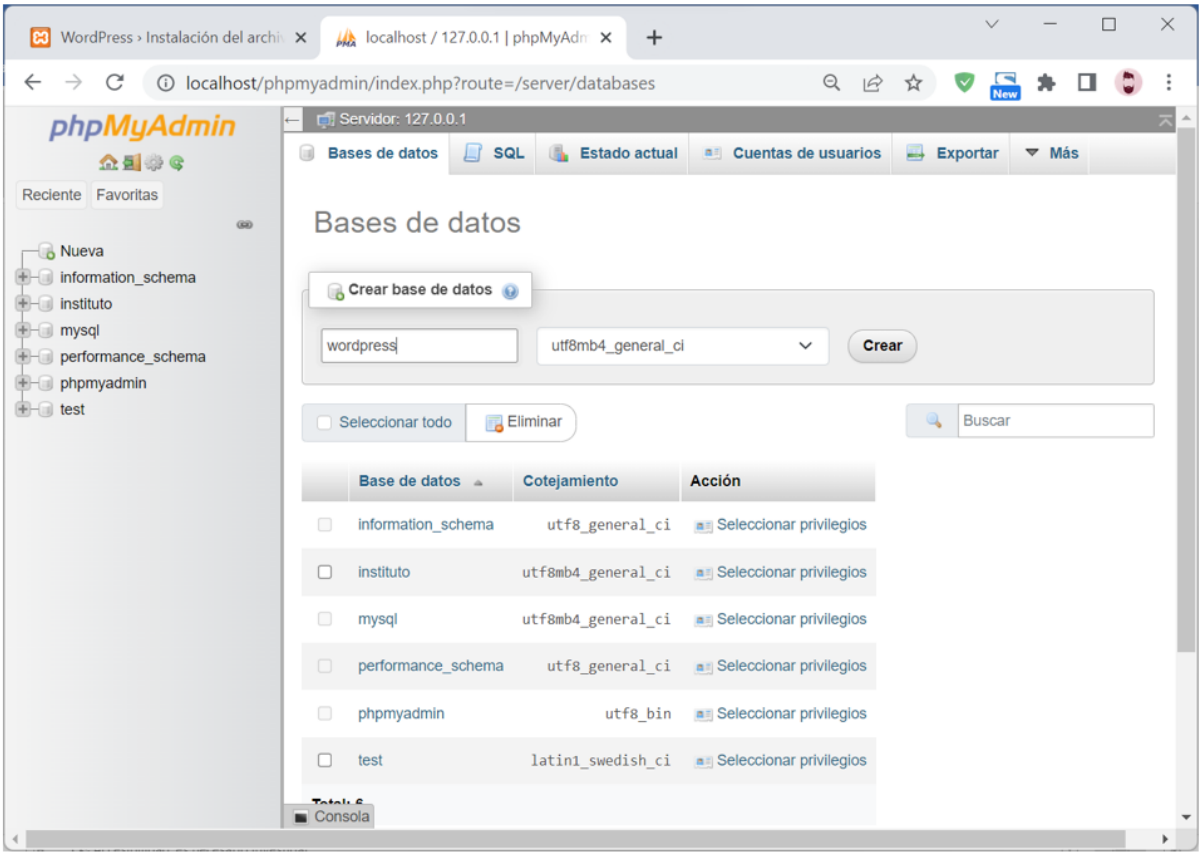
A continuación, veremos **cómo instalar WordPress**:

- **Paso 1** - Para instalar la aplicación web hace falta conseguirla y guardarla en nuestro servidor. Nosotros descargaremos WordPress de su página oficial y lo guardaremos en la carpeta pública de nuestro servidor web.

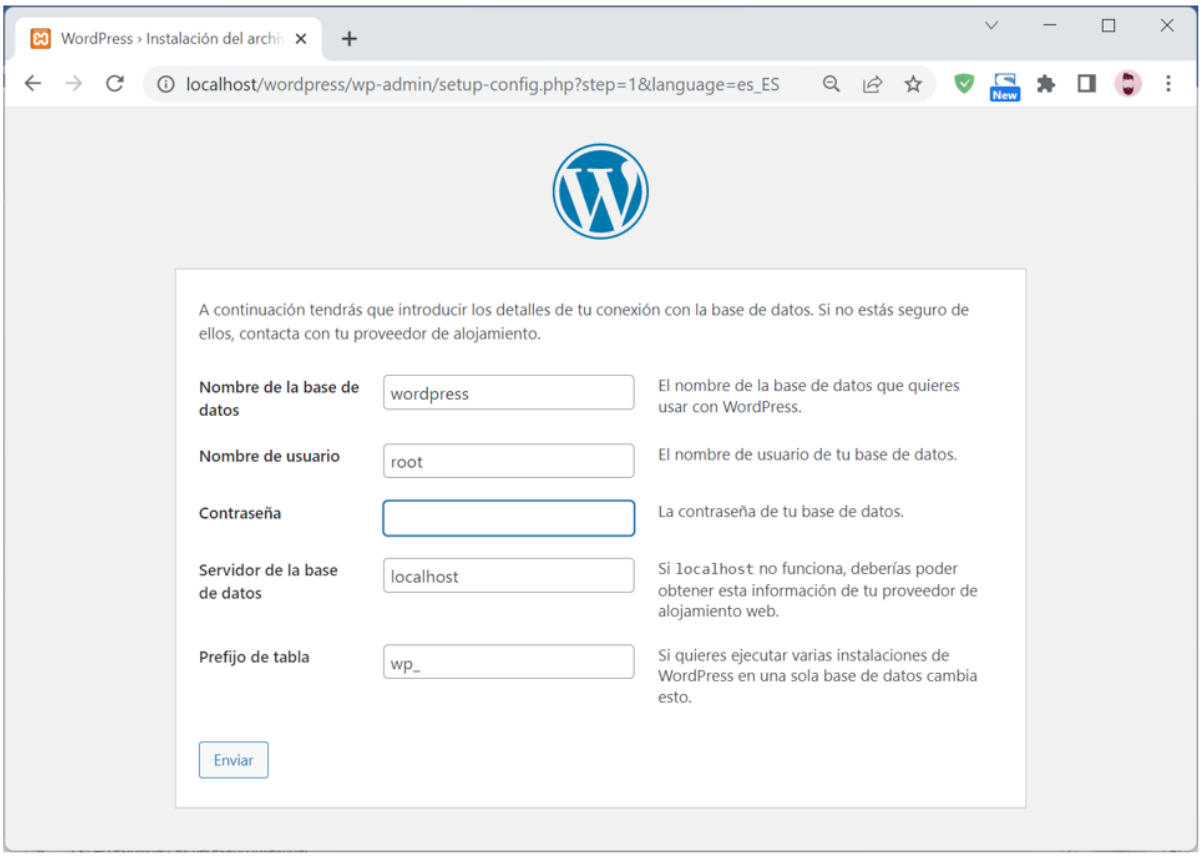
  Una vez descomprimido navegamos a su ruta, en este caso http://localhost/wordpress.



➔ **Paso 2** - Ya podemos seguir las instrucciones de instalación. Lo siguiente que haremos es ir a http://localhost/phpmyadmin y crear la base de datos para almacenar WordPress:

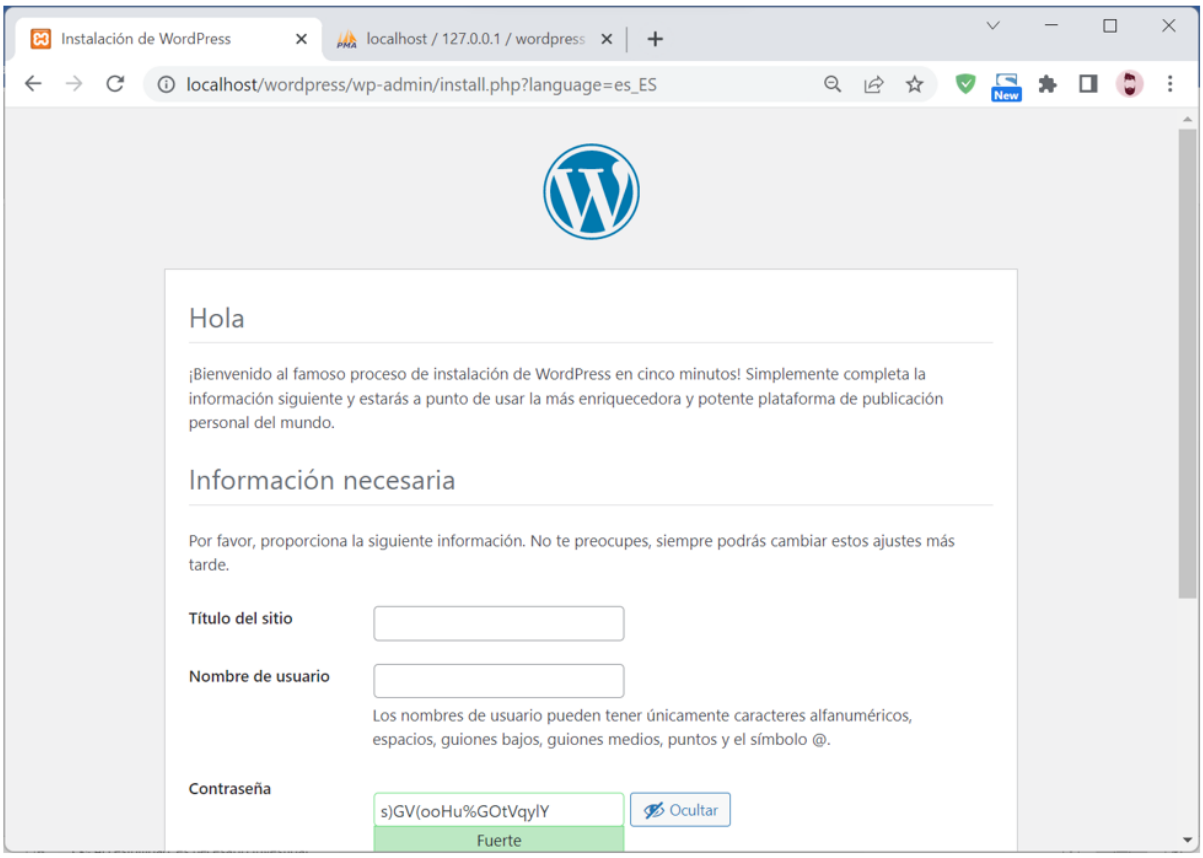


➔ **Paso 3** - Una vez creada la base de datos ya podemos rellenar los siguientes datos en la instalación de WordPress:



Asegúrate de poner los datos acorde con tu instalación de MySQL.

➔ **Paso 4** - Finalmente pulsamos sobre "ejecutar la instalación" y ya tendremos WordPress instalado, sólo falta personalizarlo a nuestro gusto.



➔ **Paso 5** - Una vez configurados nuestros datos de perfil ya tenemos un WordPress instalado y operativo:

Vídeo: instalación de WordPress

Visualiza el siguiente vídeo en el que veremos cómo se instala una aplicación web a través de WordPress.

Cuestionario

Lee el enunciado e indica la opción correcta:

Un programa bien escrito debe ser:

a. Largo.

b. Compacto.

c. Corto.

Lee el enunciado e indica la opción correcta:

En las aplicaciones web híbridas:

a. Hay que escribir el código para cada plataforma.

b. La misma aplicación servirá para todas las plataformas.

c. Sólo se repite código en el caso del sistema operativo iOS.

Lee el enunciado e indica la opción correcta:

¿Qué diferencia hay entre una aplicación web y una página web?

a. Ninguna.

b. La interactividad con los usuarios.

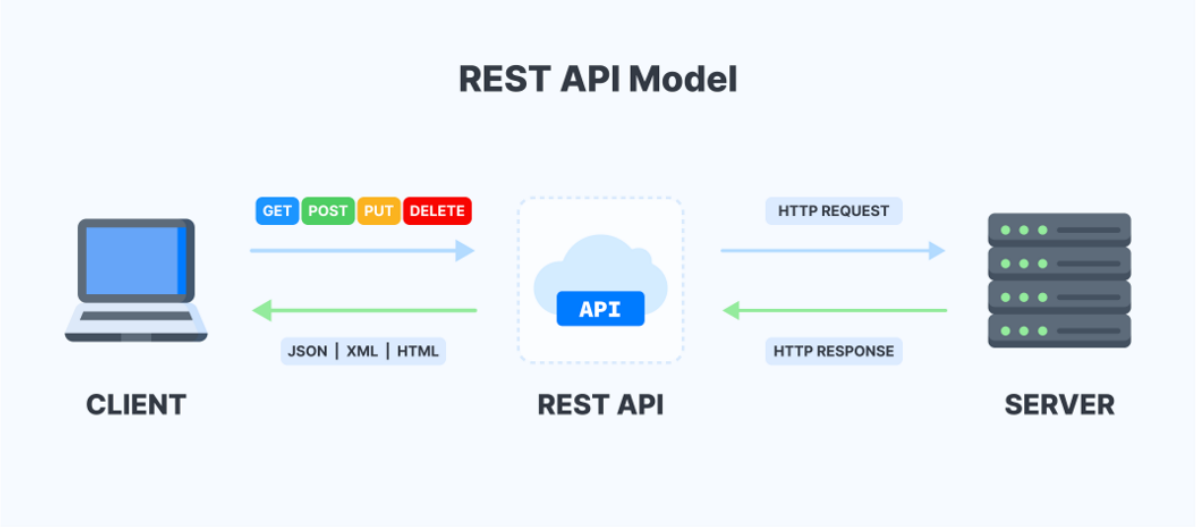
c. Que una aplicación web está escrita en PHP y una página sólo en HTML.

4. INTERFACES DE PROGRAMACIÓN

DESTACADO

*Una API es una especificación formal sobre cómo un módulo de un software interactúa con otro módulo. Dicho de otra forma, las **API** son un **conjunto de comandos, funciones y protocolos informáticos** que permiten a los desarrolladores crear programas específicos para ciertos sistemas operativos.*

Las API **simplifican** en gran parte el trabajo de un creador de programas, ya que no debe **escribir código desde cero**. Las API permiten al desarrollador utilizar funciones predefinidas para interactuar con el **sistema operativo** o con otro programa.



Funcionamiento de una API REST. Fuente: https://appmaster.io/. Esta imagen se reproduce acogiéndose al derecho de cita o reseña (art. 32 LPI), y está excluida de la licencia por defecto de estos materiales.

A continuación, veremos qué son los **REST**:

- **REST o RESTful** son un conjunto de reglas para estandarizar el uso de los webservices (peticiones HTTP hacia el servidor web).

➔ **REST nació en el año 2000**, justo un año después de SOAP, y cambió completamente la ingeniería del software. En el campo de las API, REST (Representational State Transfer, Transferencia del Estado Representacional) es, a fecha de hoy, la punta de lanza del desarrollo de servicios de aplicaciones.

➔ **Buscando otra definición** sencilla, REST es cualquier interfaz entre sistemas que utilice HTTP para obtener datos o generar operaciones sobre estos datos en todos los formatos posibles como XML o JSON.

Es una alternativa a otros protocolos estándar de intercambio de datos como SOAP (Simple Object Access Protocol), que disponen de una gran capacidad, pero también de mucha complejidad. En ocasiones es preferible una solución de manipulación más sencilla de manipulación de datos como es REST.

5. UTILIZACIÓN DE INFORMACIÓN PROVENIENTE DE REPOSITORIOS. UDDI (UNIVERSAL DESCRIPTION, DISCOVERY AND INTEGRATION)

RECUERDA

*En la unidad didáctica 8, a propósito de los webservices, ya vimos cómo recuperar información de repositorios de datos. En aquella ocasión utilizamos la API de **cdyne.com**. En la unidad didáctica 9 utilizamos la API de **TMDB** para generar contenido dinámico. Hemos utilizado estas APIs porque alguien nos ha hablado de ellas pero, ¿cuántas APIs existen? ¿Hay algún directorio o marketplace para encontrarlas? La respuesta es sí.*

Veamos las claves referentes al **uso de las API** y al uso de **datos abiertos**:

➔ **Comercialización de APIs**.

Una API puede ser comercializada como un producto, con lo cual hay tiendas que "venden APIs", por ejemplo la tienda de **RapidApi** que encuentras en recursos para ampliar.

También cualquier empresa que tenga un servicio web, es muy posible que tenga una API para subministrar datos. Un ejemplo sería una agencia meteorológica, que a parte de una posible interfaz web para que un usuario consulte el tiempo de su ciudad, puede vender datos masivos a través de una API.

También, ya hemos hablado de la **iniciativa industrial abierta** llamada UDDI que es el registro de servicios web cuyo objetivo es ser accedido por los mensajes SOAP y dar paso a documentos WSDL, en los que se describen los requisitos del protocolo y los formatos del mensaje solicitado para interactuar con dichos servicios.

➔ Otra **iniciativa interesante que tiene como eje las APIs** són los **datos abiertos**. Los datos abiertos (**open data** en inglés) son datasets producidos o recopilados por organismos públicos que las administraciones públicas ponen a disposición de la ciudadanía para que los pueda utilizar libremente de forma sencilla y cómoda.

Los datos abiertos tienen un gran valor potencial y son esenciales para la transparencia de las administraciones públicas, la eficiencia y la igualdad de oportunidades a la hora de crear riqueza.

*El **objetivo principal** de la apertura de datos es poner a disposición de la sociedad, y hacerlos públicos, los datos que gestiona la Administración, de modo que cualquier persona u organización pueda utilizarlos. Con este servicio, las administraciones aumentan la transparencia, puesto que el ciudadano accede a una visión real de la prestación de servicios. Además, la reutilización de datos abiertos por parte de empresas, entidades, asociaciones y ciudadanía en general permite elaborar nuevos productos y servicios que aporten valor, innovación, conocimiento y oportunidades de negocio.*

Estos últimos párrafos explicando qué son los datos abiertos los podrás encontrar casi con las mismas palabras en cualquier administración pública que se haya adherido a esta iniciativa.

Actividad de aprendizaje 1: reutilización de código

El project manager de tu equipo nos pasa un código que desarrolló un compañero tiempo atrás y nos comenta que a pesar de que sí funciona no es muy óptimo y toca refactorizarlo.

El código lo encontrarás en recursos para ampliar con el nombre "Actividad de aprendizaje 1 - Reutilización de código".

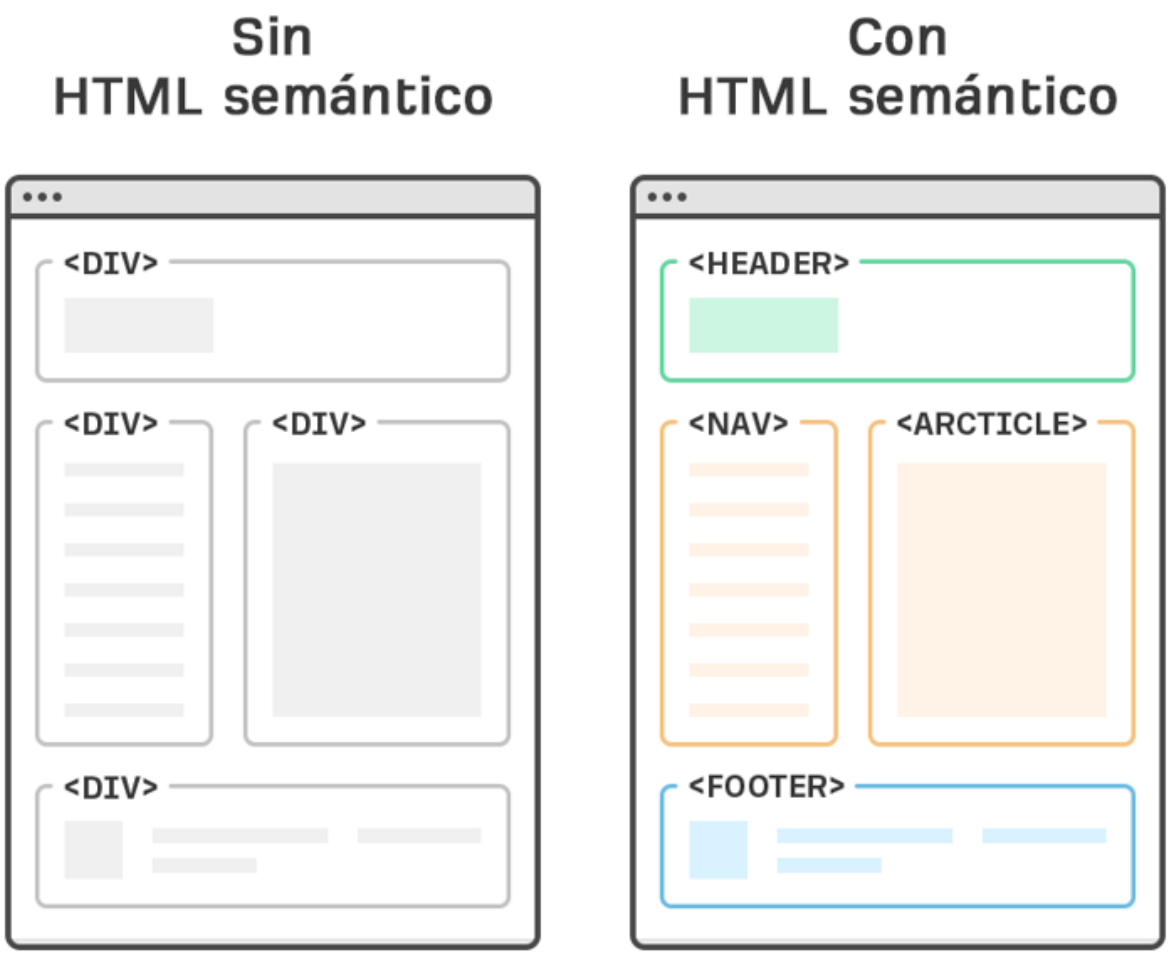
Intenta optimizarlo con lo que has visto en esta unidad didáctica sobre la reutilización de código y comparte tu estrategia en el foro del curso.

6. WEB SEMÁNTICA Y LINKED DATA

En este apartado veremos qué es la **web semántica** y **linked data**:

- Lo que conocemos como **web semántica** es un proyecto que intenta **estandarizar la forma de como publicamos los datos**. Este proyecto se basa en unas directrices que vienen marcadas por el W3C, el Consorcio WWW, en inglés: **World Wide Web Consortium**, que es un consorcio internacional que genera recomendaciones y estándares que aseguran el crecimiento de la World Wide Web a largo plazo.

Veamos un **ejemplo**. Hasta la llegada de la versión 5 de HTML los desarrolladores maquetaban una web utilizando las **etiquetas genéricas** para ello. Fíjate en la siguiente **imagen**:



Funcionamiento de una API REST. Fuente: https://appmaster.io/. Esta imagen se reproduce acogiéndose al derecho de cita o reseña (art. 32 LPI), y está excluida de la licencia por defecto de estos materiales.

Ambos diseños cumplen su función, que es meter elementos dentro de contenedores. El **HTML semántico** agrega significado al código para que las máquinas puedan reconocer bloques de navegación, encabezados, pies de página, tablas o videos.

➔ Los **linked data** o datos vinculados es la **forma como publicamos los *datasets***. Por ejemplo, si yo quiero hacer un callejero de todas las ciudades de España y cada ayuntamiento publica su callejero propio como datos abiertos,

puedo consultarlos todos, y me será más ágil gestionar todos los datos si todos los publican de la misma forma.

Siguiendo el ejemplo del callejero, si Barcelona publica el campo de código postal como **CP**, Madrid como **codigo_postal** y Sevilla como **codPost**, tendremos datos abiertos, estructurados, pero será difícil para el desarrollador trabajar con ellos.

Cuestionario

Lee el enunciado e indica la opción correcta:

¿Qué es RESTful?

a. Una característica de Laravel.

b. Un conjunto de reglas para estandarizar el uso de los webservices.

c. Un sistema operativo.

Lee el enunciado e indica la opción correcta:

¿Qué es rapidapi.com?

a. Un marketplace de APIs.

b. Un software para crear APIs.

c. Un portal de anuncios.

Lee el enunciado e indica la opción correcta:

El HTML semántico:

a. Distribuye mejor la interfaz de usuario.

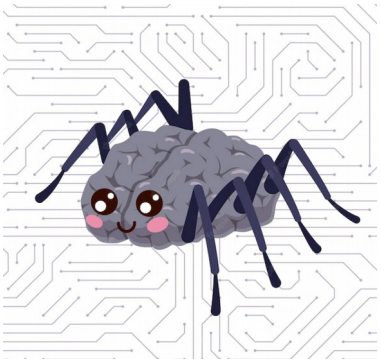
b. Es un nuevo estándar.

c. Agrega significado al código.

7. SCREEN SCRAPING

*Muchas veces queremos obtener cierta **información pública** y no hay una API para ello. Normalmente, este tipo de información es **pública** para que los usuarios la vean, pero las empresas responsables no liberan ninguna API porque quizás no interese que nadie pueda trabajar con esa información.*

Relacionado con esto, explicaremos qué es el **screen scraping** a través de un ejemplo:



- Imaginemos que tenemos una tienda online donde están nuestros productos con sus descripciones, precios e imágenes. No vamos a crear una API para obtener estos datos porque la competencia podría utilizarlos. Pero ¿y si somos la competencia y queremos utilizarlos? Entonces deberemos hacer lo que se llama **screen scraping o web scraping**, que traducido al español sería algo así como **raspado de los datos que salen por pantalla**.

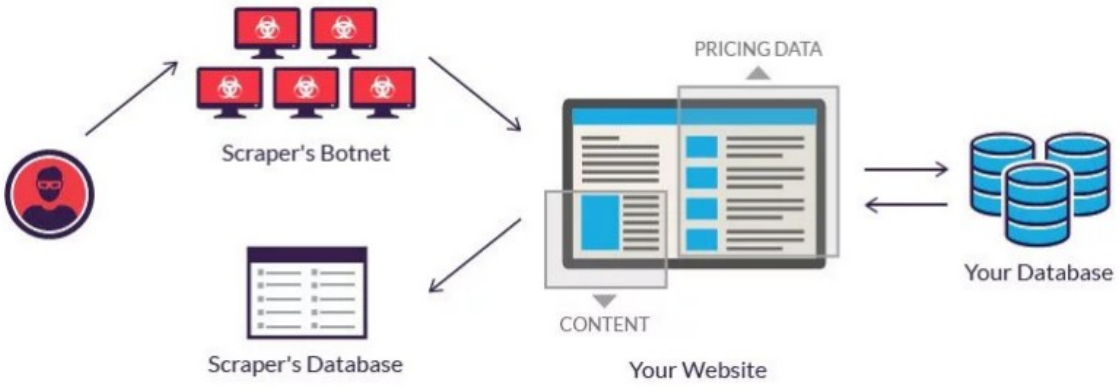


Ilustración de webscraping. Fuente: https://www.antevenio.com/. Esta imagen se reproduce acogiéndose al derecho de cita o reseña (art. 32 LPI), y está excluida de la licencia por defecto de estos materiales.

El **software programado para raspar web se le suele llamar crawler o spider**. Es como una arañita que va recorriendo web y enlaces predefinidos, o va saltando a través de ellos, recupera información y normalmente la guarda en un fichero o base de datos.

➔ Veamos un **ejemplo programado en PHP**. Imaginemos que queremos obtener todos los enlaces de los posts del blog xataka.com. Podríamos crear un pequeño script en PHP parecido al siguiente:

```
<?php
      $url = "http://www.xataka.com";
      $html = file_get_contents($url);

      preg_match_all('/<a href="http.*?">/', $html, $matches);

      print_r($matches[0]);
?>
```

Si lo probamos, veremos que ya hemos conseguido todos los enlaces. Ahora mismo, sólo **se imprimen por pantalla** y no se guardan en ningún sitio, pero podríamos almacenarlos en una **base de datos**.

Estos scripts se pueden ejecutar desde un ordenador local, desde un servidor o desde una red de servidores. Se pueden ejecutar una vez o varias veces, de forma **manual o automatizada**.

Vídeo: web scraping

Visualiza el siguiente vídeo en el que se verán un par de ejemplo de Web scraping o screen scraping.

8. CREACIÓN DE ALIMENTADORES

*Un **alimentador** es una clase especial que se utiliza para generar e insertar datos de ejemplo (semillas, o **seeders** en inglés) en una base de datos. Esta es una característica importante en los entornos de desarrollo, ya que le permite volver a crear la aplicación con una **base de datos** nueva usando valores de ejemplo que de otra manera tendría que insertar manualmente al volver a crear la base de datos.*

Como hemos trabajado con Laravel, veamos cómo este *framework* maneja los ***seeders***.

- **Paso 1**. Por defecto Laravel tiene un archivo llamado **DatabaseSeeder.php** en el directorio **/database/seeders**. Nosotros modificaremos el archivo para dejarlo tal como en la imagen, donde crearemos 10 centros aleatorios y 100 alumnos también aleatorios.

```php
<?php

namespace Database\Seeders;

use Illuminate\Database\Seeder;
use App\Models\Alumne;
use App\Models\Centre;

class DatabaseSeeder extends Seeder
{
    /**
     * Seed the application's database.
     *
     * @return void
     */
    public function run()
    {
        Centre::factory()->times(10)->create();
        Alumne::factory()->times(100)->create();
    }
}

```

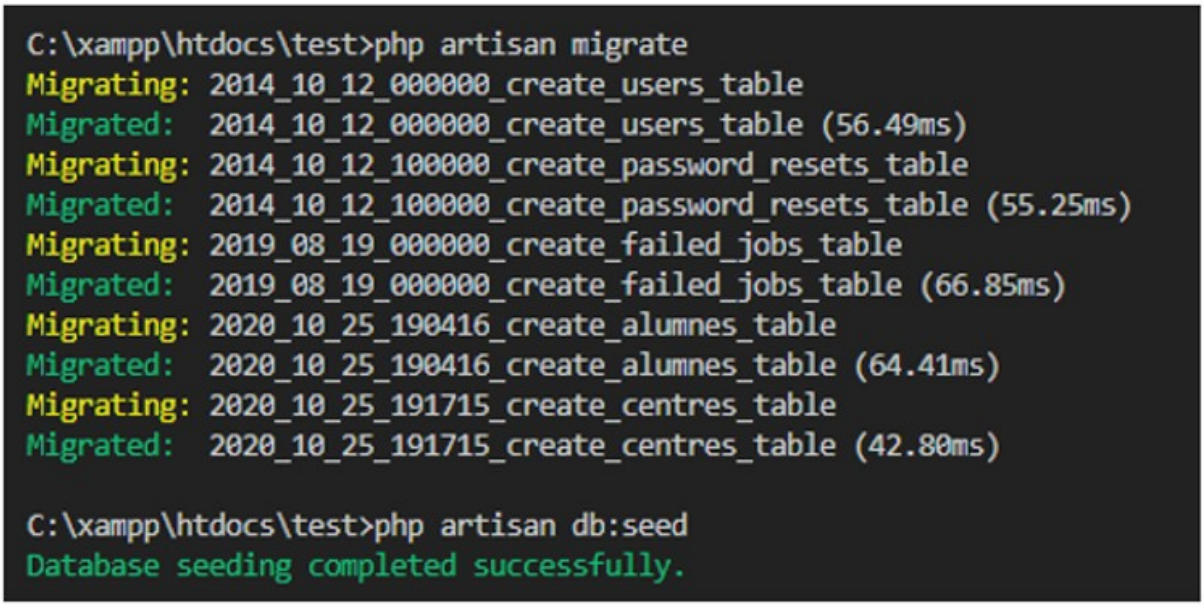
Ejemplo de creación de seeders.

No intercambies el comando si no quieres tener problemas con las foreign keys.

➔ **Paso 2**. Una vez hecho esto ya podremos lanzar el orden para llenar nuestra BD. Previamente podemos resetear la base de datos por si habíamos hecho alguna prueba:

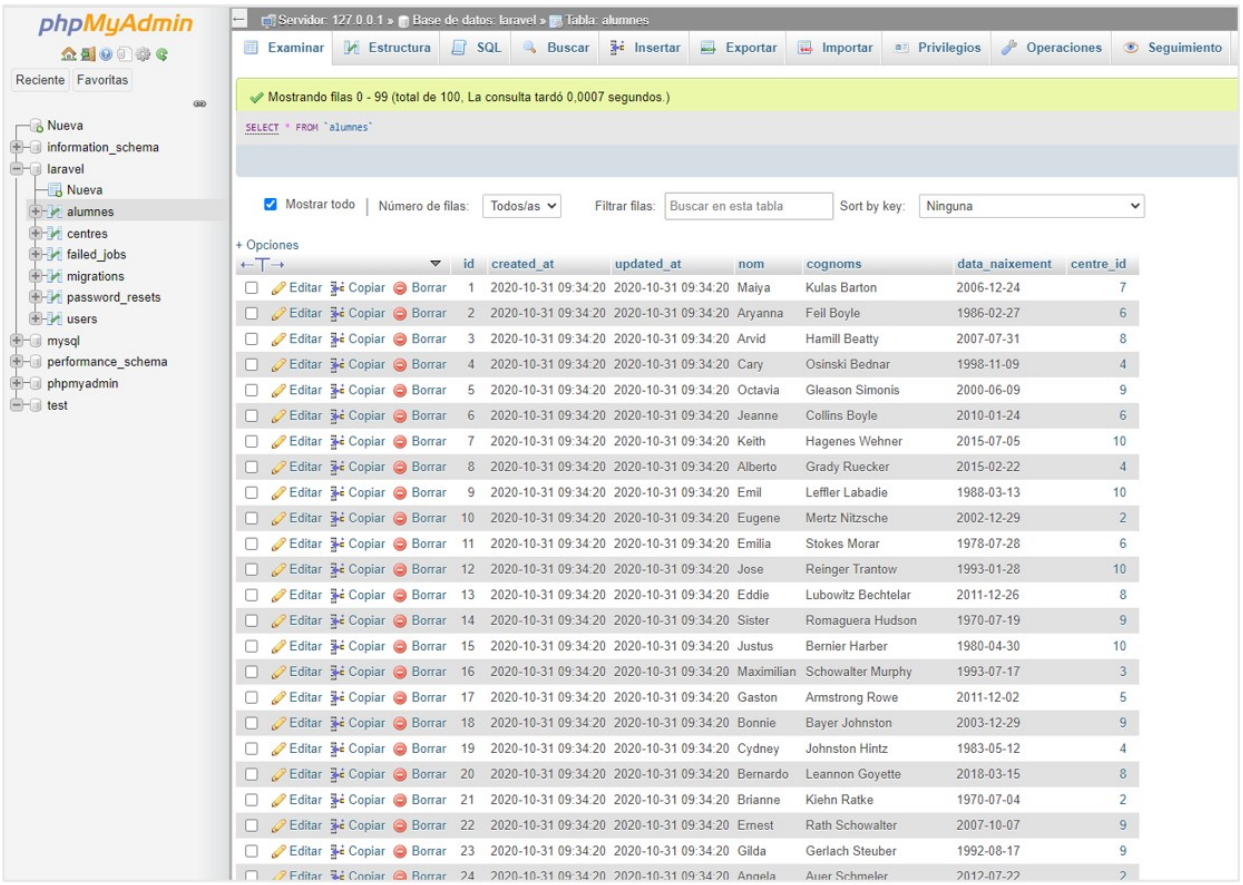
```
php artisan migrate:rollback
php artisan migrate
php artisan db:seed
```

Si todo ha funcionado correctamente veremos una imagen parecida a esta:



Mensajes de confirmación de migraciones correctas.

➔ **Paso 3**. Las migraciones se han hecho correctamente. Veamos si las tablas contemplan los datos generados aleatoriamente:



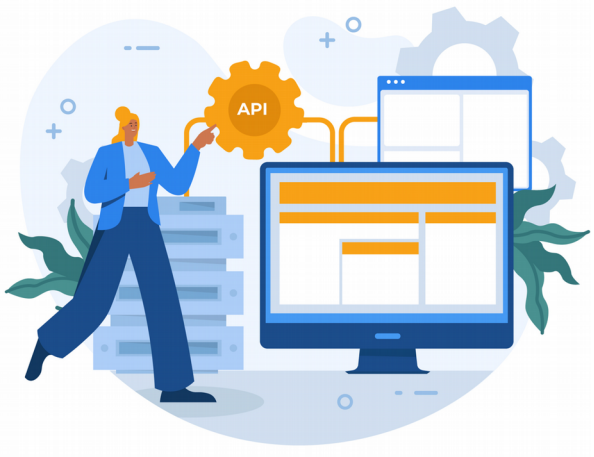
Vista de la tabla con datos aleatorios.

TOMA NOTA

Te recomendamos que visualices el vídeo "Factories y seeders" que encontrarás en recursos para ampliar.

9. CREACIÓN DE REPOSITORIOS A MEDIDA

Hasta ahora, hemos consumido APIs de terceros, pero aún no hemos **creado un repositorio propio**. Vamos a **crear** a continuación **una pequeña API** con Laravel.



➔ **Crear un nuevo proyecto**.

Primero crearemos un proyecto nuevo con Composer: `composer create-project --prefer-dist laravel/laravel restfulapi`.

El componente de rutas de Laravel nos ofrece todo lo que necesitamos para estructurar nuestra API y gracias al uso de los recursos `Route::resource()` podremos crear una colección de rutas por cada recurso en una sola línea de código: `Route::resource('user', 'UserController');`.

➔ **Crear el controlador**.

Ahora es necesario crear el controlador **UserController**. Como siempre podemos hacerlo manualmente o utilizando artisan (dentro de la carpeta del proyecto): `php artisan make:controller UserController –resource`.

Recuerda que cuando creamos una API es con el fin de que sea accesible desde otro programa, por tanto métodos como create(), edit() no son necesarios ya que sólo sirven para mostrar una vista de un formulario. Por tanto, sería mejor excluirlos de las rutas. Para ello editamos routes\api.php:

```
Route::resource('user', 'UserController',
          ['only' => ['index', 'store', 'update', 'destroy',
'show']]);
```

Fíjate que no vamos a crear ningún modelo porque utilizaremos el modelo por defecto que hay en todo proyecto de Laravel, el modelo User.

➔ **Revisar y configurar**.

Ahora es un buen momento para que revises el archivo de configuraciones (.env) y configures correctamente el acceso a tu base de datos (crea la BD que tú quieras, por ejemplo, una llamada restfulapi). A continuación, realizaremos las migraciones para que Laravel nos cree la tabla usuarios (users): `php artisan migrate`.

Ahora vamos a llenar las funciones del controlador.

➔ **Crear usuarios**.

Primero para crear usuarios será necesario programar la función **store()**:

```
public function store(Request $request)
 {
     User::create($request->all());
     return ['created' => true];
 }
```

➔ **Actualizar usuarios**.

Para actualizar usuarios será necesario programar la función **update()**:

```
public function update(Request $request, $id)
{
    $user = User::find($id);
    $user->update($request->all());
    return ['updated' => true];
}
```

➔ **Obtener datos de usuarios**.

Para obtener los datos de los usuarios será necesario programar la función **show()**:

```
public function show($id)
{
    return User::find($id);
}
```

➔ **Mostrar a los usuarios**.

También podría ser interesante que la función **index()** mostrara a todos los usuarios:

```
public function index()
{
    return User::all();
}
```

➔ **Eliminar usuarios**.

Para eliminar usuarios será necesario programar la función **destroy()**:

```
public function destroy($id)
{
    User::destroy($id);
    return ['deleted' => true];
}
```

Si nos fijamos, en todas las funciones **devolvemos un array** como respuesta. Por defecto Laravel convierte estas respuestas en objetos **JSON**.

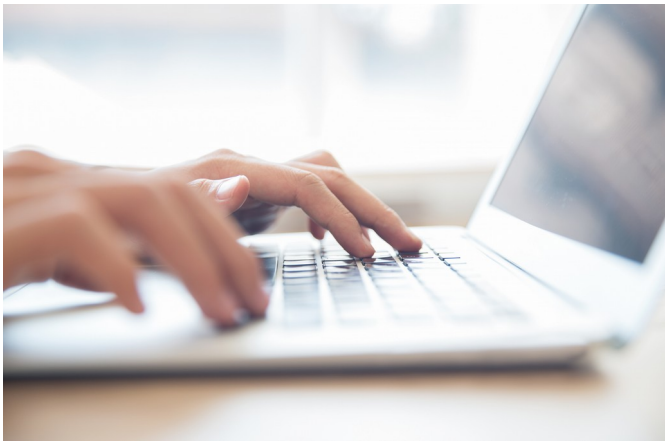
➔ **Fin del proceso**.

¡Ya hemos terminado! Probaremos si todo nuestro código funciona. Creamos manualmente un usuario con phpMyAdmin. Abrimos el navegador y hacemos una petición a nuestro servidor con el formato HTTP y la URI correcta (por ejemplo, localhost/restfulapi/public/api/user/1).

10. INCORPORACIÓN DE FUNCIONALIDADES ESPECÍFICAS

La API que hemos creado en el punto anterior es capaz de **responder a los métodos**:

- **GET**: para obtener datos de un usuario (o todos).
- **POST**: para crear un nuevo usuario.
- **PUT**: para actualizar un usuario.
- **DELETE**: para eliminar un usuario.



➔ Si un desarrollador que consuma nuestra API (o nosotros mismos) ha hecho bien el formulario de registro de un nuevo usuario no habrá tenido ningún problema. Pero **si quien consume nuestra API se equivoca**, lo ideal sería devolver un mensaje de error. Para ello aprovecharemos el validador que nos ofrece Laravel. Editar el método store():

```
public function store(Request $request)
{
      // Creaemos las reglas de validación
      $rules = [
            'name'      => 'required',
            'email'     => 'required|email',
            'password'  => 'required'
      ];

      // Ejecutamos el validador. Si falla devolvemos el error
      $validator = \Validator::make($request->all(), $rules);
      if ($validator->fails()) {
            return [
                  'created' => false,
                  'errors'  => $validator->errors()->all()
            ];
      }

      User::create($request->all());
      return ['created' => true];
}
```

➔ Otro error clásico es intentar **buscar un registro que no existe**, en nuestro caso un usuario que no existe. Tal y como hemos implementado la función show(), nuestra API mostrará una página en blanco. Estaría bien editarlo y que mostrara un error personalizado:

```
public function show($id)
 {
     return User::findOrFail($id);
 }
```

Si intentamos buscar un usuario que no existe nos saltará un error, porque ya no sólo hacemos un "find", sino que hacemos un "findOrFail".

Actividad de aprendizaje 2: creación de una API

Actividad en parejas.

Nuestra empresa se plantea crear una API que complementa la aplicación web existente. Probablemente este proyecto se nos asignará a nosotros, con lo cual, nuestro project manager nos pide practicar con una API más sencilla.

- Siguiendo los apuntes de la teoría, reprodúcelos y crea una API para listar usuarios de una base de datos. Sólo hace falta listar los usuarios, todo a la vez o a partir del ID.
- El otro miembro del equipo puede ir rellenando los campos necesarios en la base de datos (mínimo 10 usuarios).

Probad el correcto funcionamiento de la API.

11. SINDICACIÓN Y FORMATOS DE REDIFUSIÓN. RSS (RICH SITE SUMMARY)

DESTACADO

RSS *es el acrónimo de* ***Really Simple Syndication*** *o sindicación realmente simple. Una sindicación es fuente de contenidos (información estructurada) que ofrecemos a múltiples destinatarios.*

Características de RSS:

- RSS permite **sindicar el contenido** de su sitio.
- RSS define una manera fácil de **compartir y ver titulares y contenido**.
- Los archivos RSS se pueden **actualizar automáticamente**.
- RSS permite **vistas personalizadas** para diferentes sitios.
- RSS está escrito en **XML**.

Los RSS son útiles para **estructurar el contenido** de un sitio web y permitir a los usuarios que se suscriban a este contenido mediante aplicaciones que leen el formato concreto generado con XML. Si, como desarrolladores, queremos crear un RSS, lo que deberíamos hacer es **estructurar de forma manual o automática** todo el contenido de nuestra web y **guardar el archivo XML** en una ruta, por ejemplo `/feed`. Un feed muy famoso es el del blog de WordPress.

Código: estructura del archivo XML para RSS

```
<?xml version="1.0" encoding="UTF-8" ?>
<rss version="2.0">

<channel>
  <title>Ejemplo</title>
  <link>https://www.ejemplo.com</link>
  <description>Descripción del canal ejemplo</description>
  <item>
    <title>Entrada 1</title>
    <link> https://www.ejemplo.com/entrada1.php</link>
    <description>Ejemplo entrada 1</description>
  </item>
  <item>
    <title>Entrada 2</title>
    <link> https://www.ejemplo.com/entrada2.php</link>
    <description>Ejemplo entrada 2</description>
  </item></channel>

</rss>
```

Encontrarás más información sobre las etiquetas RSS en la bibliografía.

Actividad de aprendizaje 3: creación de RSS

La página web de nuestra empresa, a pesar de estar muy bien diseñada, no está sindicada. Recursos humanos, que se encargan de la comunicación, nos piden sindicar el blog de la empresa.

Crea de forma manual un RSS para tus posts. Puedes inventarte el contenido, pero debe haber un mínimo de 3 entradas.

Cuando hayas terminado valida tu documento.

Cuestionario

Lee el enunciado e indica la opción correcta:

Cuando no tenemos una API y queremos conseguir los recursos de una web utilizamos:

a. Web scraping.

b. Web soling.

c. Deep web.

Lee el enunciado e indica la opción correcta:

Un alimentador es:

a. Un generador de datos aleatorios.

b. Un generador de código automático.

c. Una pieza que da energía a los servidores.

Lee el enunciado e indica la opción correcta:

¿Qué funciones no son necesarias cuando creamos una API?

a. Show() y destroy().

b. Create() y edit().

c. Store() y update().

RESUMEN

Esta unidad didáctica ha sido la guinda del pastel de todo el módulo. En ella hemos repasado y puesto en práctica los conocimientos sobre **APIs** vistos en unidades anteriores. También hemos visto cómo **obtener datos sin necesidad de una API** scrapeando una web.

También hemos creado **código limpio** reutilizando nuestro propio código y encapsulándolo en bloques lógicos. El código limpio en sí ya es una buena práctica y si además tenemos que crear webs híbridas que se adapten a distintos dispositivos hace que sea algo indispensable.

También hemos practicado instalando un **CMS** en nuestro servidor. Se ha aprovechado la conjetura y el CMS en cuestión ha sido WordPress que a su vez tiene aplicaciones para móviles.

RECURSOS PARA AMPLIAR

PÁGINAS WEB

- Actividad de aprendizaje 1 - Reutilización de código: *https://imagenes.contenidosteleformacion.es/ampliar/ense2021/unidad10/actividad_%20reutilizacion_codigo.pdf* [Consulta noviembre 2022].
- Construir un feed RSS con Laravel: *https://devdojo.com/bobbyiliev/how-to-add-a-simple-rss-feed-to-laravel-without-using-package* [Consulta noviembre 2022].
- Construir una API segura con Laravel: *https://auth0.com/blog/build-and-secure-laravel-api/* [Consulta noviembre 2022].
- Factories y seeders: *https://www.youtube.com/watch?v=cG2Ok7AWDUk* [Consulta noviembre 2022].
- Página oficial WordPress: *https://es.wordpress.org/download/* [Consulta noviembre 2022].
- Página web para validar documentos: *https://validator.w3.org/feed/#validate_by_input* [Consulta noviembre 2022].
- Tutorial de POO en PHP de w3schools: *https://www.w3schools.com/php/php_oop_what_is.asp* [Consulta noviembre 2022].
- Web de RapidApi: *https://rapidapi.com/* [Consulta noviembre 2022].

BIBLIOGRAFÍA

PÁGINAS WEB

- Documentación de WordPress: *https://wordpress.org/support/* [Consulta noviembre 2022].
- Refencia de seeders en Laravel: *https://laravel.com/docs/9.x/seeding* [Consulta noviembre 2022].
- Referencia oficial de PHP: *https://www.php.net/* [Consulta noviembre 2022].
- Tutorial de RSS de w3schools: *https://www.w3schools.com/xml/xml_rss.asp* [Consulta noviembre 2022].

GLOSARIO

- **API**: siglas de application programing interface. Las API son un conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizada por otro software como una capa de abstracción.
- **CMS**: siglas de content management system, o gestor de contenidos.
- **UDDI**: siglas de Universal Description, Discovery and Integration, o catálogo de webservices.
- **UI**: siglas de user interface, interfaz de usuario. Es el medio con que el usuario puede comunicarse con una máquina y comprende todos los puntos de contacto entre el usuario y el equipo.
- **UX**: siglas de user experience, experiencia de usuario. Es el conjunto de factores y elementos relativos a la interacción del usuario con un entorno o dispositivo concretos, dando como resultado una percepción positiva o negativa de dicho servicio, producto o dispositivo.
- **W3C**: siglas de World Wide Web Consortium. Es un consorcio internacional que genera recomendaciones y estándares que aseguran el crecimiento de la World Wide Web a largo plazo.