

GamesAPI

API de videojuegos

Valentín Castravete – 3º DAM DAW DUAL

Índice

Enlaces de acceso al contenido y despliegue del mismo.....	3
Enlaces de herramientas utilizadas para el desarrollo	3
Introducción.....	4
Objetivos	7
Análisis.....	7
Planificación	10
Diseño	11
Sprints	12
Primer mes	12
Último mes.....	12
Implantación	13
Composer y bramus/router	13
Conexión a base de datos y actualización de la misma	15
Llamadas API	16
Objetos.....	18
Puesta en producción	25
Conclusiones.....	27

Enlaces de acceso al contenido y despliegue del mismo

GitHub: <https://github.com/valentincastravete/GamesAPI>

Página web: <https://valentin.ociobinario.com/>

Enlaces de herramientas utilizadas para el desarrollo

bramus/router: <https://github.com/bramus/router>

Composer: <https://getcomposer.org/>

Introducción

GamesAPI trata sobre una **API REST pública y abierta de videojuegos**.

Hablaré durante todo el proyecto acerca de un videojuego en concreto, que es el primero de la API, Monster Hunter World. Esto porque en el futuro se crearían más para diferentes videojuegos.



Intenciones y público objetivo

La información que ofrece la API está orientada a los jugadores principalmente del propio videojuego del que se ofrece la información y/u organizaciones que quieran realizar una **wiki** acerca del videojuego, es decir, maquetarla para ofrecerla de una forma visual al cliente final, que serían los jugadores de los propios videojuegos, los cuáles acudirían a esta por ayuda o intereses personales acerca del videojuego.

Un ejemplo de wiki de este videojuego es [fextralife](#).

Aparte de la propia API, también habrá una página web que ofrecerá, mediante un fácil filtrado, el enlace por el cual hay que pedir a la API una información en concreto. Este filtrado sería variable en base al videojuego del que se ofrece la información.

Será un formulario el cuál presente desde un desplegable todas las entidades por las que buscar información.

He querido realizar este proyecto porque me gusta el hecho de facilitar la información a los demás. Además de querer mejorar las API que ya existen y unificándolas en una página (dominio/videojuego en el futuro para evitar alcanzar un límite de llamadas a la API).

Todo el proceso se basa en:

- Primero descargar una base de datos actualizada del repositorio de GitHub del que voy a obtener los datos.
- Al ser una base de datos SQLite, es muy sencilla de manejar.
- Procesar todas las entidades juntando y modificando tablas de información necesaria para otras.
- Devolverlo en formato JSON.
- Todas las URL disponibles se pueden ver en la página de inicio de la página web gracias a un formulario dinámico con diferentes valores a seleccionar, así como el juego, idioma, objetos, etc. Todo esto variable dependiendo del videojuego.
- Además, la base de datos se mantiene actualizada gracias a un cron creado en el servidor, el cuál descarga de la última versión del proyecto de GitHub la base de datos SQLite.

API

Una **API** es un conjunto de código de programación que permite la transmisión de datos entre un producto de software y otro. También contiene los términos de este intercambio de datos.

Pública: También conocidas como orientadas al desarrollador o externas, estas API están disponibles para cualquier desarrollador de terceros. Un programa de API pública permite aumentar el conocimiento de la marca y recibir una fuente adicional de ingresos cuando se ejecuta correctamente.

Las **API públicas abiertas** son gratuitas. La definición de API abierta sugiere que la descripción de la API y cualquier documentación relacionada deben estar disponibles abiertamente y que la API se puede utilizar libremente para crear y probar aplicaciones.

Hay diferentes **tipos de usos para las API**, como la API **remota**, de **sistemas operativos**, **web** y **bases de datos**.

El de este proyecto está orientado a una **API de base de datos**, que permiten la comunicación entre una aplicación y un sistema de administración de bases de datos. Los desarrolladores trabajan con bases de datos escribiendo consultas para acceder a datos, cambiar tablas, etc.

Y según el **protocolo** que usa, es **REST**, es decir, cada recurso está representado por una URL única, y se puede solicitar este recurso proporcionando su URL.

La información devuelta se encuentra en JSON, que contiene colecciones de pares nombre/valor y listas ordenadas de valores. Dado que se trata de estructuras de datos universales, el formato se puede utilizar con cualquier lenguaje de programación.

Videojuegos

Los **videojuegos** son juegos digitales, jugados en plataformas como un ordenador, consolas, móviles, etc. Estos requieren de una interfaz de usuario y/o un periférico de entrada como un mando, teclado, etc., para generar una respuesta. Esta respuesta es mostrada en un dispositivo de visualización como una televisión, monitor, gafas de realidad virtual, etc. Algunos de estos videojuegos incluyen sonidos y/o música para mejorar la experiencia de usuario.

Los videojuegos se pueden **clasificar gracias en base a estas características:**

Género: Es la principal característica que define un videojuego. Define el estilo del mismo en base al modo de juego. Algunos de estos son juegos de disparos, de terror, fantasía, medieval, etc.

Modo: Esto especifica cuántos jugadores pueden jugar, en qué momentos, desde qué perspectivas y el orden entre todo ello. Los principales son un solo jugador y multijugador.

Intenciones: Indica la orientación, finalidad, emociones o sensaciones a generar en el usuario final definidas por la forma de jugar el mismo, aunque bajo mi punto de vista, esto sigue dependiendo en una pequeña parte de como el usuario final quiera jugar el videojuego, pero sin esto cambiar las intenciones iniciales del creador del videojuego.

En el caso del **primer videojuego del que va a ofrecer información mi proyecto** es **Monster Hunter World**.

Es un videojuego con **género de acción y mundo abierto, modo multijugador y cooperativo**. Pretende unir tanto la gran historia que tiene detrás como la gran y amplia acción que ofrece, además del juego entre otros jugadores que lo hace más enriquecedor.

Una **wiki** es una página web en la cual se documenta en profundidad cualquier dato, información, forma de jugar, etc., acerca de un tema en concreto (en nuestro caso para los videojuegos).

Objetivos

Estos son todos los objetivos que busco obtener al realizar este proyecto:

- Ofrecer una API REST pública y abierta de uno o varios videojuegos.
- Mantener los datos ofrecidos actualizados periódicamente.
- Facilitar a las personas o entidades que hagan uso de mi API la creación de documentación acerca de uno y/o varios videojuegos.
- Ofrecer una página web que ofrezca URLs a la API diferentes en base a una configuración seleccionada por el usuario final.
- Aprender a desarrollar una API con una librería de php orientada exclusivamente a este tipo de API.
- Realizar un correcto desarrollo y mejorar en todos los aspectos relacionados a programar.
- Finalizar el curso actual para poder proseguir con mi desarrollo personal y profesional.

Análisis

Proyectos similares

Tras buscar detenidamente varias API que ofrezcan este mismo tipo de información, me he encontrado con este [proyecto](#) que me ha llamado la atención, llamado mhw-db, el cual ofrece exactamente la misma información que el proyecto que estoy realizando.

Según mi búsqueda, este ha sido el mejor proyecto con una misma finalidad que el mío, ya que incluso es utilizado para una aplicación de móvil para mostrar datos del videojuego en cuestión.

El resto de proyectos encontrados son proyectos incompletos, falta de información o no internacionalización.

Pero la diferencia principal entre este proyecto y el mío, es que mhw-db ofrece las URL de información explicadas en una web con documentación de ello, pero no expone todas las URL disponibles. Lo que mi proyecto si tendrá será una página web principal con unos filtros y desplegados indicando la URL de los recursos o recurso que el usuario esté buscando, facilitándole la búsqueda.

Tras buscar de donde ofrecía este proyecto su información, he encontrado este proyecto de github con información estructurada del videojuego:

[MHWorldData](#)

El cuál ofrece en archivos csv, un archivo parecido a un archivo Excel, la información de casi todo el videojuego.

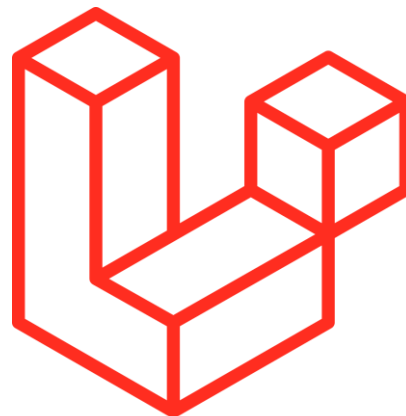
Como he comentado anteriormente, el proyecto de MHWorldData ofrece un archivo de base de datos, lo cual ahorrara mucho tiempo.

Lo complicado de todo esto es relacionar la información de las decenas de tablas (47 en concreto) entre algunas de ellas para ofrecer una información más útil en una única llamada.

Como personalmente he jugado al videojuego durante muchas horas y de forma técnica, se exactamente cuáles son las tablas e información necesaria a relacionar en una única llamada.

Modo de desarrollo y herramientas a utilizar

Al comenzar con todo esto, lo primero en lo que pensé fue en usar un framework conocido como Symfony o Laravel, orientados a desarrollo de proyectos con diferentes orientaciones, entre estas, una API.



Pero tras leer y analizar, me di cuenta de que ninguna de las dos se adaptaba del todo al propósito y tamaño de mi proyecto. Ya que Symfony está orientado a proyectos muy grandes, como PrestaShop, y que Laravel, aunque sí que está orientado a proyectos más pequeños, sigue siendo muy complejo para la sencillez que yo busco en mi proyecto.

Gracias a uno de los jefes de la empresa en la que estoy cursando mi formación en centro de trabajo, me ha recomendado una **librería de puro PHP** que facilita mucho la creación de una API, con simples funciones por llamada o URL, sin definir ningún controlador y/o modelo. Únicamente hay que definir las rutas y el archivo de acceso a la base de datos.

Con esto, he podido crear rápidamente llamadas a los objetos de la base de datos. La parte complicada es relacionar los datos de las tablas y modificar su disposición para una lectura más cómoda.

La librería en cuestión se llama [bramus/router](#).

Para instalar esta librería hay que hacer uso de [Composer](#), un sistema de gestión de paquetes para programar en PHP.

Con este, descargamos todos los paquetes PHP de los que requiere esta librería, incluyendo la propia librería.



Para la gestión y acceso a la base de datos, he elegido SQLite ya que tiene un formato muy cómodo ya que es muy sencillo de manejar.

Para hacer uso de este tipo de base de datos con el objeto SQLite3, hay que habilitar su uso en el archivo php.ini.

Planificación

Mi planificación no ha sido plasmada en ningún sitio.

He pensado en lo necesario dependiendo del punto en el que se encuentre el proyecto.

Empecé realizando la memoria con unas primeras pinceladas de la idea de mi proyecto, y eso ocurrió durante un par de horas, las cuáles no quedaron muy claras para mí, como mi tutor del proyecto me comentó, por lo tanto, dediqué unas horas más de investigación de otros proyectos, finalidad, herramientas de desarrollo, etc.

Gracias a esa investigación, pude seguir adelante mejorando mis ideas respecto al proyecto en la memoria.

Tras haber tenido las ideas claras, pasé primero a la configuración inicial de mi proyecto, que fue configurar la librería que iba a utilizar con Composer, durando este un par de horas ya que tenía que entender como funcionaba tanto la librería como Composer.

Se configura el archivo composer.json, el cuál contiene los paquetes y dependencias que Composer instalará, en este caso solo se incluye la propia librería de bramus/router.

Después de una hora de documentación de la librería y el gestor de paquetes y de la configuración, pasé a desarrollar directamente.

Comencé por la conexión a la base de datos SQLite. Y además de esto, cree un archivo PHP para actualizar la base de datos, y con un cron en el servidor, este se ejecutaría diariamente para mantenerla actualizada. Todo esto en unas 2 horas de desarrollo incluyendo documentación del mismo.

Antes de desarrollar las propias llamadas, hay que configurar la librería con un par de parámetros, esto unos pocos minutos.

En el desarrollo de las llamadas no había una posible planificación como tal, ya que, al ser una API, todas las llamadas son iguales, lo único que cambia es la forma de modificar y relacionar los datos de diferentes tablas. Esta parte ha sido la que más tiempo ha requerido, han sido alrededor de 30 horas de desarrollo puro para mejorar la información a devolver por llamada.

A medida que creaba más llamadas, veía que mucho código se duplicaba, por lo tanto, optimicé el código lo máximo posible. Esto también incluye en las 30 horas anteriores, pero de estas, 5 horas han sido para optimizar, ya que tenía que también parametrizar las funciones para cada objeto.

Por último, pasé a desarrollar la página de inicio, desde la cual, mediante un formulario dinámico (pero no la información), muestra la URL a utilizar para el recurso que se requiera. Esto ha sido más rápido, 3 horas, ya que hice uso de una plantilla HTML para el formulario, uno simple y limpio.

Diseño

El diseño en mi proyecto solo incluye la página en la que ofrezco las diferentes URL.

El diseño inicial que tenía en mente es algo simple, moderno y limpio, y es al final lo que he conseguido. No he realizado mockup porque he considerado que, al querer un diseño tan simple, no habría cambio entre versiones.

Este es el resultado, con el cual he quedado agradecido:



The screenshot shows a web application titled "GamesAPI". It features a form with the following elements:

- URL:** A text input field containing the URL "https://valentin.ociobinario.com/monster_hunter_world/monsters". To the right of the input is a button labeled "Abrir".
- Juego:** A dropdown menu with "Monster Hunter World" selected.
- Idioma:** An empty dropdown menu.
- Objeto:** A dropdown menu with "Monsters" selected.

Sprints

Dividiendo toda la planificación realizada en sprints, quedaría así:

Primer mes

Pinceladas de la idea principal del proyecto.

Último mes

Tras reuniones con el tutor, investigo más.

Con la investigación, avanzo la parte de introducción, objetivos y análisis de la memoria en base a esta.

Busco herramientas de desarrollo a utilizar para realizar un proyecto simple y rápido.

Configuración inicial de las herramientas en el proyecto.

Desarrollo de acceso a la base de datos y actualización de la misma.

Desarrollo de todas las llamadas a los diferentes objetos teniendo en cuenta sus relaciones con otras tablas.

Mejora y limpieza del código.

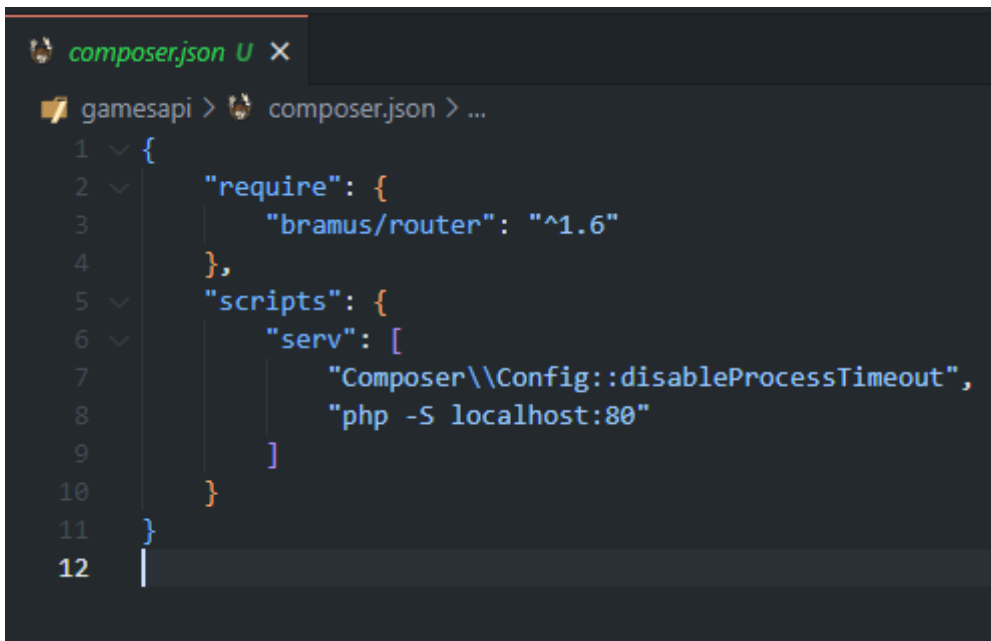
Desarrollo de la página de inicio con el formulario para filtrar/generar las URL para el usuario.

Implantación

Este apartado incluye la misma información que los demás, pero más detallado a nivel de código.

Composer y bramus/router

Para este punto, es necesario crear un archivo incluyendo las dependencias, en este caso, bramus/router:



```
1 {  
2     "require": {  
3         "bramus/router": "^1.6"  
4     },  
5     "scripts": {  
6         "serv": [  
7             "Composer\\Config::disableProcessTimeout",  
8             "php -S localhost:80"  
9         ]  
10    }  
11 }  
12
```

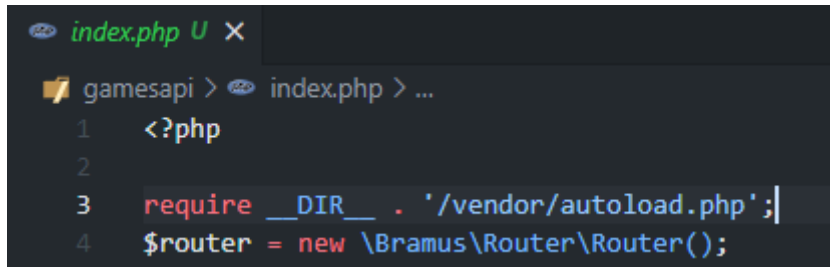
Con el require se indican estas dependencias.

En el apartado de scripts, he creado uno personal para poder ejecutar el proyecto en segundo plano para localhost.

Este script se ejecuta con el comando:

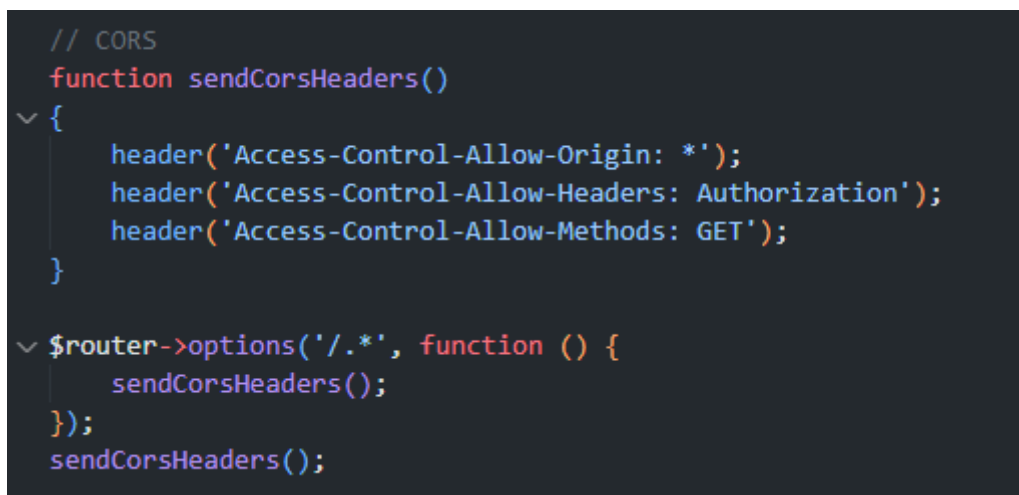
composer 'nombre_servicio(serv en este caso)'

En el archivo PHP principal, incluimos el autoload para cargar los enrutamientos y creamos el objeto router de nuestra librería.



```
index.php U X
gamesapi > index.php > ...
1  <?php
2
3  require __DIR__ . '/vendor/autoload.php';
4  $router = new \Bramus\Router\Router();
```

Y definimos las opciones CORS (permitir protocolos específicos para cada llamada, en nuestro caso solo los GET ya que es una API REST):



```
// CORS
function sendCorsHeaders()
{
    header('Access-Control-Allow-Origin: *');
    header('Access-Control-Allow-Headers: Authorization');
    header('Access-Control-Allow-Methods: GET');
}

$router->options('/*.*', function () {
    sendCorsHeaders();
});
sendCorsHeaders();
```

Conexión a base de datos y actualización de la misma

Antes de poder utilizar el objeto SQLite3 en php, hay que activar este en el archivo php.ini descomentando estas 2 líneas:

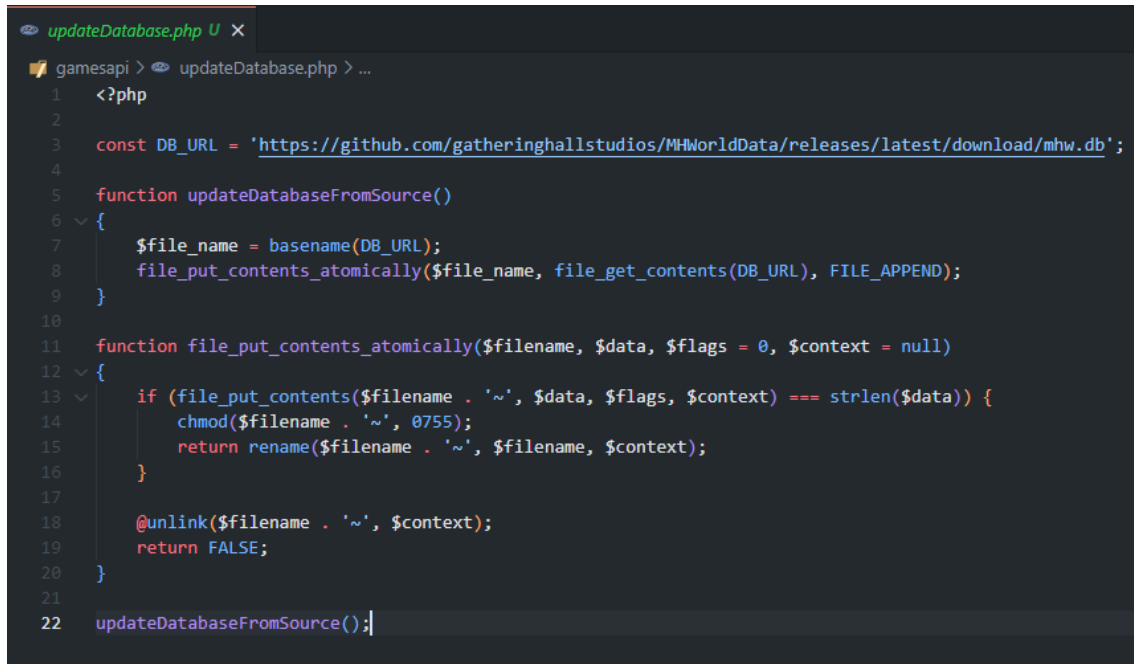
```
936 ;extension=pdo_oci
937 ;extension=pdo_pgsql
938 extension=pdo_sqlite
939 ;extension=pgsql
940 ;extension=shmop
941
942 ; The MIBS data available in the
943 ; See https://www.php.net/manual/
944 ;extension=snmp
945
946 ;extension=soap
947 ;extension=sockets
948 ;extension=sodium
949 extension=sqlite3
950 ;extension=tidy
951 ;extension=xsl
952
```

Para la conexión he creado una clase específica. No es necesario especificar usuario y contraseña para realizar la conexión.

```
class SQLiteConnection
{
    /**
     * SQLite3 instance
     * @var SQLite3
     */
    private static $bd;

    /**
     * return in instance of the PDO object that connects to the SQLite database
     * @return \SQLite3
     */
    public static function getDB()
    {
        if (self::$bd == null) {
            try {
                if (!file_exists(PATH_TO_SQLITE_FILE)) {
                    throw new Exception();
                }
                self::$bd = new SQLite3(PATH_TO_SQLITE_FILE);
            } catch (PDOException $ex) {
                header('Content-Type: application/json; charset=utf-8');
                echo json_encode(['error' => 500, 'message' => 'Connection to Database Failed.']);
            } catch (Exception $e) {
                header('Content-Type: application/json; charset=utf-8');
                echo json_encode(['error' => 500, 'message' => 'Connection to Database Failed. Database file not found.']);
            }
        }
        return self::$bd;
    }
}
```

Para actualizar la base de datos, he creado este archivo PHP:

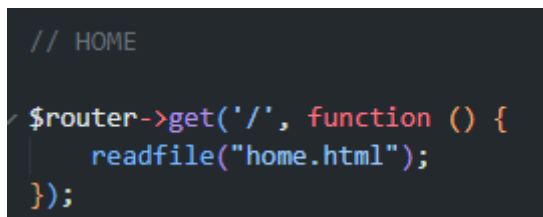


```
1 <?php
2
3 const DB_URL = 'https://github.com/gatheringhallstudios/MHWorldData/releases/latest/download/mhw.db';
4
5 function updateDatabaseFromSource()
6 {
7     $file_name = basename(DB_URL);
8     file_put_contents_atomically($file_name, file_get_contents(DB_URL), FILE_APPEND);
9 }
10
11 function file_put_contents_atomically($filename, $data, $flags = 0, $context = null)
12 {
13     if (file_put_contents($filename . '~', $data, $flags, $context) === strlen($data)) {
14         chmod($filename . '~', 0755);
15         return rename($filename . '~', $filename, $context);
16     }
17
18     @unlink($filename . '~', $context);
19     return FALSE;
20 }
21
22 updateDatabaseFromSource();
```

El cuál busca en los últimos lanzamientos del proyecto de GitHub.

Llamadas API

Cada una de estas llamadas son así de simples:



```
// HOME
$router->get('/', function () {
    readfile("home.html");
});
```

Se llama a la función get del objeto router, incluimos la ruta que el usuario tiene que escribir y la función devolviendo la información que necesitamos, en este caso es la página de inicio, por lo cual hago una redirección a su archivo HTML correspondiente.

También es posible montar llamadas, es decir, incluir una llamada en las de su interior:

```
$router->mount('/monster_hunter_world', function () use ($db, $router, $lang) {  
  
    // MONSTERS  
    $router->get('/([a-z]{2}/)?monsters', function ($language = null) use ($db, $lang) {
```

En este caso se incluye siempre en la ruta URL/monster_hunter_world/...

En la definición de la URL de la que el usuario tiene que hacer uso también se le pueden definir expresiones regulares:

```
// MONSTERS  
$router->get('/([a-z]{2}/)?monsters', function ($language = null) use ($db, $lang) {  
    header('Content-Type: application/json');
```

Aquí estoy indicando que se pueden incluir opcionalmente 2 letras en minúsculas como un parámetro más, siendo este para el idioma.

Estos son varios ejemplos funcionales:

URL/monster_hunter_world/monsters

URL/monster_hunter_world/es/monsters

En cada llamada indicamos que vamos a devolver contenido JSON:

```
header('Content-Type: application/json');
```

Objetos

Paso a explicar la llamada a los monstruos como ejemplo.

```
// MONSTERS
$router->get('/([a-z]{2}/)?monsters', function ($language = null) use ($db, $lang) {
    header('Content-Type: application/json');

    $lang = $language != null ? $language : $lang;

    $extra_monster_info = getExtraMonsterInfo($db, $lang);
    $monsters_text = $extra_monster_info[0];
    $monsters_locations = $extra_monster_info[1];
    $locations_text = $extra_monster_info[2];
    $monsters_rewards = $extra_monster_info[3];
    $monsters_rewards_conditions_text = $extra_monster_info[4];
    $items = $extra_monster_info[5];
    $items_text = $extra_monster_info[6];

    // Get monsters
    $monsters_result = $db->query("SELECT * FROM monster ORDER BY order_id");
    $monsters = [];
    if ($monsters_result && $monsters_result->numColumns() > 0) {
        while ($monster = $monsters_result->fetchArray(SQLITE3_ASSOC)) {
            $monster = proccessMonster(
                $monster,
                $monsters_text,
                $monsters_locations,
                $locations_text,
                $monsters_rewards,
                $monsters_rewards_conditions_text,
                $items,
                $items_text
            );
            $monsters[] = $monster;
        }
    }

    returnJsonData($monsters);
});
```

En la primera línea establezco el idioma según si el usuario lo ha indicado o no, si no, hago uso del idioma en inglés por defecto.

En las siguientes líneas obtengo mediante sentencias SQL la información de todas las tablas necesarias para devolver un objeto con toda su información.

Esto es lo que obtengo con esas líneas:

```
function getExtraMonsterInfo(SQLite3 $db, string $lang): array
{
    // Get monsters text
    $monsters_text = executeSQL($db, "SELECT * FROM monster_text WHERE lang_id = '" . $lang . "'");
    // Get monsters locations
    $monsters_locations = executeSQL($db, "SELECT * FROM monster_habitat");
    // Get locations text
    $locations_text = executeSQL($db, "SELECT id, name FROM location_text WHERE lang_id = '" . $lang . "'");
    // Get monsters rewards
    $monsters_rewards = executeSQL($db, "SELECT * FROM monster_reward");
    // Get rewards conditions text
    $monsters_rewards_conditions_text = executeSQL($db, "SELECT id, name FROM monster_reward_condition_text WHERE lang_id = '" . $lang . "'");
    // Get items
    $items = executeSQL($db, "SELECT * FROM item");
    // Get items text
    $items_text = executeSQL($db, "SELECT id, name, description FROM item_text WHERE lang_id = '" . $lang . "'");

    return [
        $monsters_text,
        $monsters_locations,
        $locations_text,
        $monsters_rewards,
        $monsters_rewards_conditions_text,
        $items,
        $items_text
    ];
}
```

En la sentencia siguiente simplemente obtengo todos los objetos de la base de datos y después los recorro para procesar cada uno individualmente, tras procesarlos, los asigno a un array y luego hago uso de json_encode para devolverlo.

Esta es la función para procesar cada objeto:

```
function processMonster(
    array $monster,
    array $monsters_text,
    array $monsters_locations,
    array $locations_text,
    array $monsters_rewards,
    array $monsters_rewards_conditions_text,
    array $items,
    array $items_text
): array {
    if (!empty($monsters_text)) {
        $monster_info = insertAdditionalInfoToMonster($monster, $monsters_text);
        $monster = $monster_info[0];
        $monster_text = $monster_info[1];
    }

    $result_fields = splitMonsterFields($monster);
    $monster = $result_fields[0];
    $removed_fields = $result_fields[1];

    $monster['traps'] = setMonsterTraps($removed_fields);
    $monster['ailments'] = setMonsterAilments($removed_fields);
    $monster['weaknesses'] = setMonsterWeaknesses($removed_fields, $monster_text);

    if (!empty($monsters_locations) && !empty($locations_text)) {
        $monster['locations'] = processMonsterLocations($monster, $monsters_locations, $locations_text);
    }
    if (!empty($monsters_rewards) && !empty($monsters_rewards_conditions_text)) {
        $monster['rewards'] = processMonstersRewards($monster, $monsters_rewards, $monsters_rewards_conditions_text, $items, $items_text);
    }
    return $monster;
}
```

Para empezar, el objeto no contiene el nombre y otros datos en su propia tabla, por lo tanto, tengo que acceder a la tabla de nombres por idioma y añadir esos campos:

```
function insertAdditionalInfoToMonster(array $monster, array $monsters_text): array
{
    // Get text of monster
    $monster_text = getRecordFromValues($monster, $monsters_text, 'id', 'id', true);
    // Insert additional info to monster (name, description, ecology)
    return [insertInfoToArray($monster, $monster_text, 'size', ['name', 'description', 'ecology']), $monster_text];
}
```

Aquí podemos ver que obtengo los textos con su idioma y después añado esos campos al objeto.

Más abajo en el código de procesar el objeto, divido en 2 el objeto ya que una parte se puede segmentar mucho más y la otra contiene datos únicos:

```
$result_fields = splitMonsterFields($monster);
$monster = $result_fields[0];
$removed_fields = $result_fields[1];
```

Esto se divide según un campo:

```
function splitMonsterFields(array $monster): array
{
    //Extract and remove monster fields
    return splitArray($monster, 'pitfall_trap', ['order_id']);
}
```

Esta función también elimina campos que no se necesiten.

Y en las siguientes líneas:

```
$monster['traps'] = setMonsterTraps($removed_fields);  
$monster['ailments'] = setMonsterAilments($removed_fields);  
$monster['weaknesses'] = setMonsterWeaknesses($removed_fields, $monster_text);
```

Todos esos campos divididos para segmentar y agrupar, aquí se procesan por separado, quedando el resultado así:

```
▼ traps:  
  pitfall:      1  
  shock:        1  
  vine:         1  
▼ ailments:  
  roar:         "small"  
  bleed:        1  
▼ weaknesses:  
  fire:         1  
  water:        1  
  ice:          3  
  thunder:      2  
  dragon:       0  
  poison:       1  
  sleep:        2  
  paralysis:    3  
  blast:        2  
  stun:         2
```

Y esto es lo que queda en la parte de datos únicos y los campos añadidos de idioma(japonés):

```
id:      30  
name:    "オドガロン"  
▼ description:  
ecology: "Fanged Wyvern"  
size:    "large"  
icon:    null
```

He notado que en la base de datos faltan muchas traducciones.

Para terminar, procesamos otros datos utilizando varias tablas:

```
if (!empty($monsters_locations) && !empty($locations_text)) {  
    $monster['locations'] = processMonsterLocations($monster, $monsters_locations, $locations_text);  
}  
if (!empty($monsters_rewards) && !empty($monsters_rewards_conditions_text)) {  
    $monster['rewards'] = processMonstersRewards($monster, $monsters_rewards, $monsters_rewards_conditions_text, $items, $items_text);  
}
```

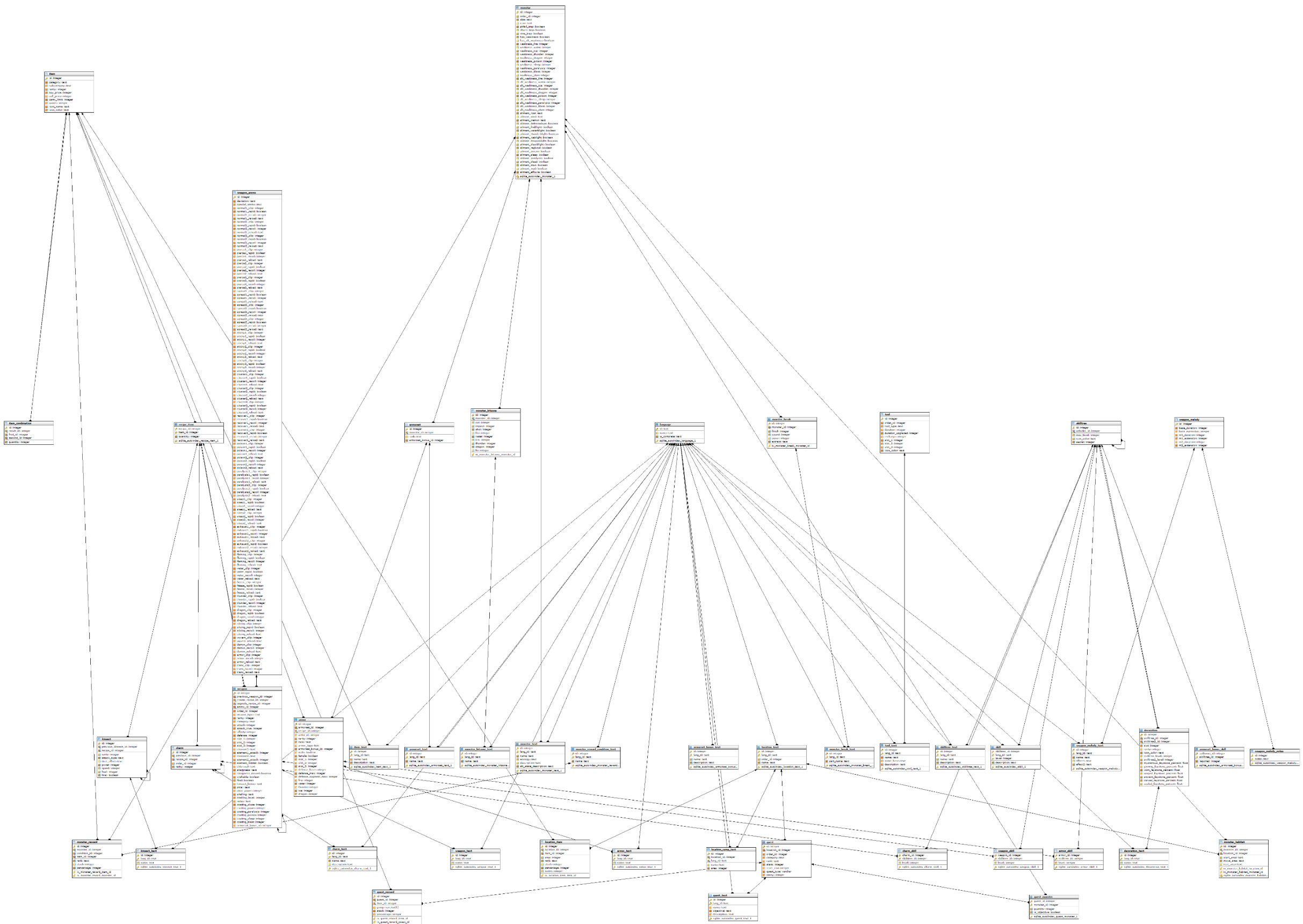
Quedando el resultado así:

```
▼ locations:  
  ▼ 0:  
    id: 3  
    name: "陸珊瑚の台地"  
    start_area: null  
    move_area: "2,4,5,8"  
    rest_area: "4"  
  ▼ 1:  
    id: 4  
    name: "瘴気の谷"  
    start_area: null  
    move_area: "3,6,8,9,10,13"  
    rest_area: "13"  
▼ rewards:  
  ▼ 0:  
    id: 1962  
    ▼ item:  
      name: "惨爪竜の鱗"  
      description: "オドガロンの素材。主に剥ぎ取りで入手できる。汎用性が高く、幅広い用途に使われる。"  
      condition: "痕跡採取"  
      rank: "LR"  
      stack: 1  
      percentage: 100  
  ▼ 1:  
    id: 1963  
    ▼ item:  
      name: "惨爪竜の鱗"  
      description: "オドガロンの素材。主に剥ぎ取りで入手できる。汎用性が高く、幅広い用途に使われる。"  
      condition: "Carve / Capture"  
      rank: "LR"  
      stack: 1  
      percentage: 35  
  ▼ 2:  
    id: 1964  
    ▼ item:  
      name: "惨爪竜の硬筋"  
      description: "オドガロンの素材。主に報酬で入手できる。硬い材質で、防具によく使われる。"  
      condition: "Carve / Capture"  
      rank: "LR"  
      stack: 1  
      percentage: 25  
  ▼ 3:
```

Este es otro ejemplo, pero de otro objeto, que ha requerido de más procesos que el objeto anterior:

id:	30
name:	"Brazales de Jagras"
type:	"arms"
rarity:	1
rank:	"LR"
armorset:	
id:	8
name:	"Jagras"
pieces:	
27:	
id:	28
name:	"Yelmo de Jagras"
type:	"head"
28:	
id:	29
name:	"Cota de Jagras"
type:	"chest"
29:	
id:	30
name:	"Brazales de Jagras"
type:	"arms"
30:	
id:	31
name:	"Faja de Jagras"
type:	"waist"
31:	
id:	32
name:	"Greas de Jagras"
type:	"legs"
skills:	
0:	
id:	85
name:	"Mejora de camarada"
description:	"Fortalece a los camaradas."
level:	1
max_level:	5

Este es el esquema entidad relación de la base de datos del videojuego del que GamesAPI ofrece información:

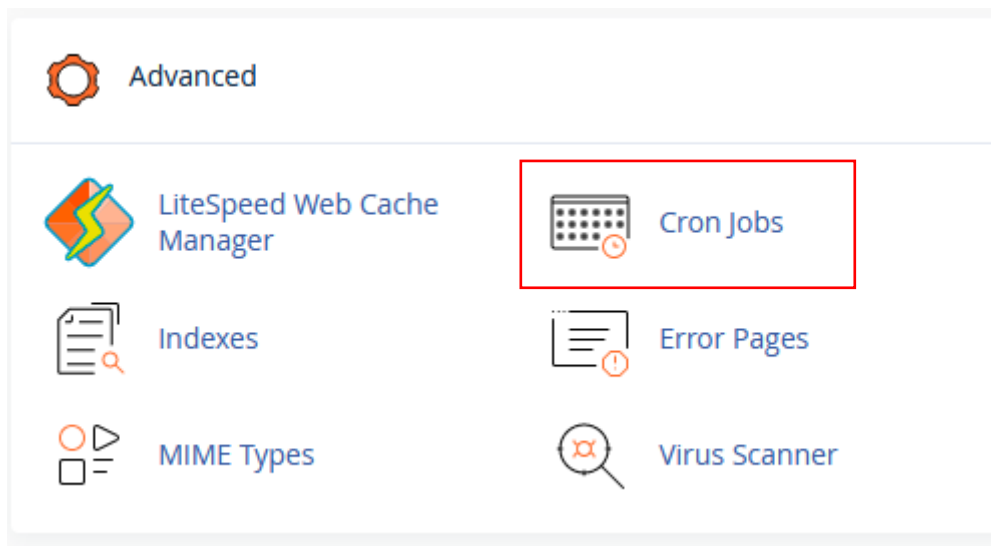


Puesta en producción

El alojamiento web utilizado para mi proyecto ha sido el ya usado para el anterior proyecto Logrocho, proporcionado por uno de nuestros profesores.

[webuphosting](#)

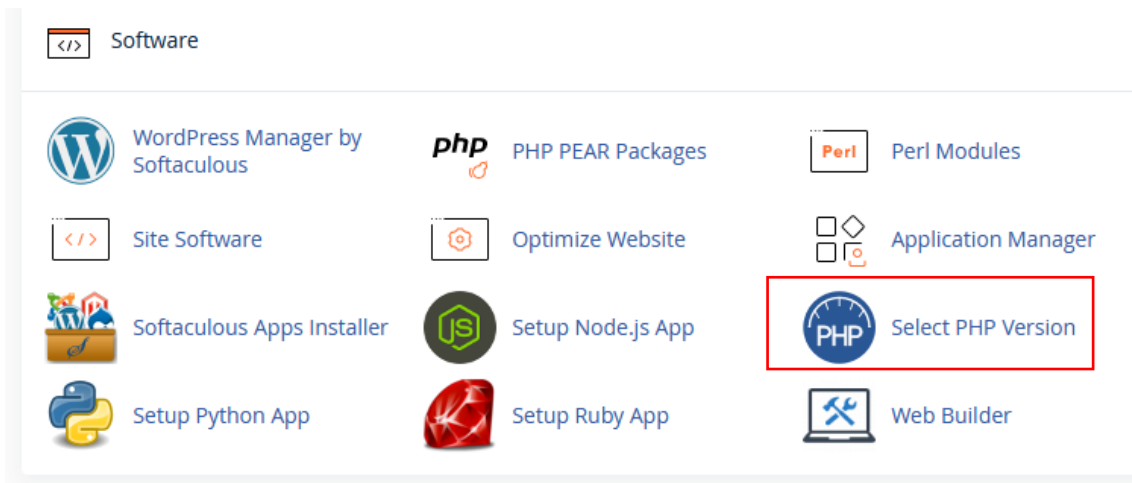
Desde aquí, en el panel del backend, se encuentran las crons del servidor, en la cual he creado la actualización de la base de datos periódica:



Aquí encontramos la cron job creada para tal acción:

Current Cron Jobs							Actions	
Minute	Hour	Day	Month	Weekday	Command			
0	0	*	*	0	<code>@ * * * * /usr/local/bin/php /home/valentinociobina/public_html/updateDatabase.php</code>		Edit	Delete

Para la activación de la extensión SQLite3 en este entorno, he aprovechado un cambio anterior de versión de PHP ya que el código no lo podía interpretar con una versión antigua y desde aquí se pudo habilitar tal extensión:



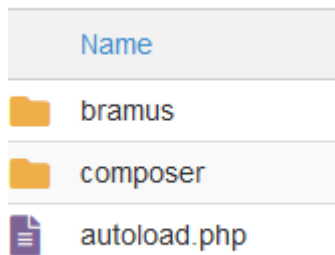
Cambiada la versión de 7.1 a 8.1, también al haber cambiado la versión, ha habilitado la modificación de extensiones PHP:

PHP Extensions

Current PHP version: **8.1 (current)**

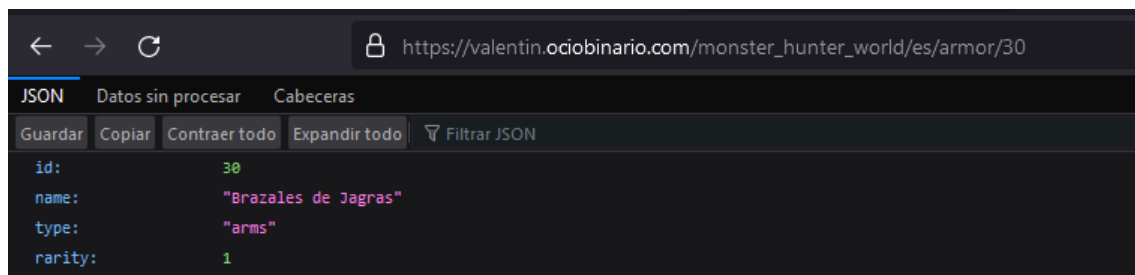
B <input checked="" type="checkbox"/> bcmath	M <input checked="" type="checkbox"/> mbstring	P <input checked="" type="checkbox"/> pdo_mysql	S <input checked="" type="checkbox"/> soap
D <input type="checkbox"/> dba	<input type="checkbox"/> mysqli	<input type="checkbox"/> pdo_oci	<input checked="" type="checkbox"/> sockets
<input checked="" type="checkbox"/> dom	<input checked="" type="checkbox"/> mysqlind	<input type="checkbox"/> pdo_odbc	<input type="checkbox"/> sodium
E <input type="checkbox"/> enchant	N <input checked="" type="checkbox"/> nd_mysql	<input type="checkbox"/> pdo_pgsql	<input checked="" type="checkbox"/> sqlite3
F <input type="checkbox"/> ffi	<input type="checkbox"/> nd_pdo_mysql	<input checked="" type="checkbox"/> pdo_sqlite	<input type="checkbox"/> sysvmsg
<input checked="" type="checkbox"/> fileinfo	O <input type="checkbox"/> odbc	<input type="checkbox"/> pgsql	<input type="checkbox"/> sysvsem
G <input checked="" type="checkbox"/> gd	<input checked="" type="checkbox"/> opcache	<input checked="" type="checkbox"/> phar	<input type="checkbox"/> sysvshm
I <input checked="" type="checkbox"/> imap	P <input checked="" type="checkbox"/> pdo	<input checked="" type="checkbox"/> posix	T <input type="checkbox"/> tidy
<input checked="" type="checkbox"/> intl	<input type="checkbox"/> pdo_dblib	<input type="checkbox"/> pspell	X <input checked="" type="checkbox"/> xmlreader
L <input type="checkbox"/> ldap	<input type="checkbox"/> pdo_firebird	S <input type="checkbox"/> snmp	<input checked="" type="checkbox"/> xmlwriter
X <input checked="" type="checkbox"/> xsl			
Z <input checked="" type="checkbox"/> zip			

Para el composer, no he podido instalarlo, por lo tanto, he simplemente usado FTP para subir los archivos que ha creado la instalación en mi local y funciona perfectamente.



Y tras cada desarrollo, actualizaba los archivos con FTP y probaba en “producción”.

Ya tenemos nuestra API REST GamesAPI en producción, un alojamiento web.



Conclusiones

Mis primeras impresiones de este proyecto son que me gusta mucho programar y darle vueltas a la cabeza, además de procesar datos de un tema que a mí personalmente me encanta, que son los videojuegos.

De hecho, pensaba que al realizar este tipo de proyecto iba a acercarme algo al desarrollo de videojuegos, pero como mi tutor del proyecto me comentó en mis objetivos, eso no pasará, ya que son desarrollos completamente diferentes. Pero por lo menos sí me he dado cuenta que trabajo con más entusiasmo cuando el tema es algo que a cada uno nos guste.

Me he dado cuenta, gracias a este proyecto y haber procrastinado tanto con él, de que, tengo un problema personal que me afecta en todos los aspectos de la vida y espero que, con el tiempo y ayuda, esto cambie y encuentre una motivación real para realizar cualquier proyecto u objetivo en la vida.

Si es verdad que no he desarrollado ningún plan de planificación o sprints previo ni de ningún aspecto debido al tema comentado anteriormente.

De lo que me alegro es de haber podido realizar un desarrollo con una herramienta nueva para mí y como he comentado antes, respecto a un tema que me gusta.