

Desarrollo de una página web

Comparación de Pokemons

Servicios y aplicaciones en Red

30 - Noviembre - 2016

*Mikel Fernandez
Sergio Tobal*

Índice

1. Objetivos	2
1.1 Estructura	2
2. Funcionamiento	4
2.1 Puntos a mejorar	8
3. Tecnologías usadas	10
3.1 HTML5 & CSS3	10
3.2 JavaScript & jQuery & AJAX	10
3.2.1 jQuery Accordion	10
3.2.2 JavaScript SessionStorage	12
3.3 PHP	14
3.4 Servicios web - API REST	15
3.4 Hosting	17
4. Conclusiones	17
5. Referencias	18

1. Objetivos

El objetivo de la práctica es el desarrollo de una página web haciendo uso de las tecnologías HTML, CSS, XML, JavaScript y PHP. Para llevar esto a cabo, hemos pensado primero en una idea general del proyecto, para después plantear el diseño de una web que incluya el uso de todas las tecnologías solicitadas.

La idea de la mencionada web ha surgido al escuchar a varios compañeros discutir sobre el uso de bases de datos para esta práctica. Se comentaba que no era lógico crear una base de datos con una cantidad elevada de elementos con todos sus atributos. A raíz de ello se ha pensado que, efectivamente, no es necesario crear a mano una base de datos y, además, que ni siquiera es necesario disponer de una base de datos local. Esto se consigue mediante el uso de una API que nos proporciona la información que le solicitamos en tiempo real, y nos permite crear una web que gestione una gran cantidad de datos y permite al usuario acceder a ellos mediante una interfaz sencilla e intuitiva. Tras una búsqueda simple se ha encontrado la página web *pokeapi.co*, la cual nos proporciona los datos que necesitamos a través de una API REST. De esta forma evitamos tener que introducir los datos a mano y disponemos de una cantidad de datos que, incluso tomándose la molestia de crear nuestra propia base de datos, no es probable que alcanzáramos dichas dimensiones.

Una vez fijado el uso de la API y tomando la misma como idea central para la web, se ha procedido a plantear el resto. Surge la idea de comparar 2 de los elementos recogidos mediante la API según sus datos. También se plantea la opción de que el usuario tenga un espacio donde introducir sus notas sobre dichas comparaciones (o cualquier otro apunte que vea necesario). Además, se piensa en un sencillo sistema de comentarios que los usuarios de la página quieran dejar acerca de la misma. Para que la visualización de los comentarios no requiera cambios ni refrescos de página y se hagan en tiempo real, introducimos el uso de la tecnología AJAX para este tipo de funcionamientos.

Habiendo fijado ya estas características de nuestra web, queda definida la idea básica: realización de comparaciones entre pares de Pokemons. También queda definido su funcionamiento y, por último, las tecnologías que van a usarse en cada apartado del sistema.

1.1 Estructura

Inicialmente se optó por una estructura avanzada usando la popular y práctica plantilla *HTML5 Boilerplate*, que proporciona los ficheros de documentación, una configuración básica para control de errores, control de estilos, páginas de error personalizadas, scripts para funciones básicas de retrocompatibilidad y más detalles para tener una página web robusta y completa según los estándares actuales.

No obstante, por simplicidad se ha acabado optando por una estructura simple con carpetas separadas para el css y el javascript, así como los siguientes ficheros:

- **Composer.lock** y **Composer.json**: Ya que en nuestro servidor se hace uso de *Composer* para gestionar dependencias y paquetes, requerimos de sus archivos de configuración. El archivo JSON lo creamos manualmente y se usa principalmente para definir los *plugins* que tiene que gestionar y el archivo con extensión *.lock* es un archivo que genera *Composer* al ejecutarlo por primera vez con la configuración introducida en nuestro json.
- **Humans.txt** y **Robots.txt**: Pertenecen al compendio de ficheros *HTML5 Boilerplate* y los hemos decidido mantener, ya que nos parecían un buen añadido; *robots.txt* ayuda a los bots que visitan nuestra web a saber donde pueden y donde no pueden entrar, por si no queremos evitar la entrada de algún robot a la web total o parcialmente. *humans.txt*, muestra en un texto plano toda la información sobre los creadores de la web.
- **Procfile**: Es un archivo solicitado por Heroku, que por detrás tiene un gestor de dependencias llamado Foreman, que se encarga de ver las instancias o *workers* solicitados en el fichero y crearlos en nuestro repositorio de Heroku. Es decir, nos sirve para definir si vamos a tener una web simple, si vamos a necesitar plugins de bases de datos, o cuáles servicios va a tener exactamente nuestro proyecto.
- **Favicon**: Al ejecutar el servidor PHP siempre nos saltaba un error de que faltaba un *favicon.ico*, así que decidimos añadirlo. Este fichero tiene la función de diferenciar la pestaña de nuestra web respecto de otras que tenga el usuario abierto en su navegador cambiando el icono que aparece junto al título de la pestaña.



Fig.1 La pestaña de la página web muestra, junto a Title, el favicon de la misma

- **Validación**: Los navegadores pueden interpretar el código HTML escrito de muchas maneras, pero se puede validar el código para comprobar que es correcto según un estándar. Como hemos hecho uso de la versión 5 de HTML hemos usado la web *html5.validator.nu* para realizar la validación. La parte de PHP la hemos intentado validar con *phpcodechecker.com*, pero parece ser algo laxo con las correcciones. Por último, el XML se ha validado y comprobado que está bien formateado creando para él un DTD que hemos incluido al inicio del fichero XML.

(X)HTML5 validation results for http://polar-savannah-45299.herokuapp.com

Validator Input

Address

☐ Show Image Report

☐ Show Source

The document is valid HTML5 + ARIA + SVG 1.1 + MathML 2.0 (subject to the utter previewness of this service).

Used the HTML parser. Externally specified character encoding was UTF-8.

Total execution time 109 milliseconds.

[About this Service](#) • [More options](#)

Fig.2 Aspecto del validador de HTML5

2. Funcionamiento

El servicio consta de 4 apartados, 2 de ellos pensados para comparar datos y las 2 restantes para almacenar información.

Notes Select Compare

En el siguiente bloque de texto podras guardar tus notas, o los resultados de tus comparaciones

Escribe aqui tus notas, o las comparaciones recibidas

Comentarios acerca de PokeCompare:

nombre (16:28:23, 11 December, 2016)
come1

nomh2 (16:29:23, 11 December, 2016)
com2

Visitante (20:37:20, 12 December, 2016)
Comentario sobre la pagina Web

Añade tu propio comentario con AJAX:

Nombre:

E-mail:

Comentario:

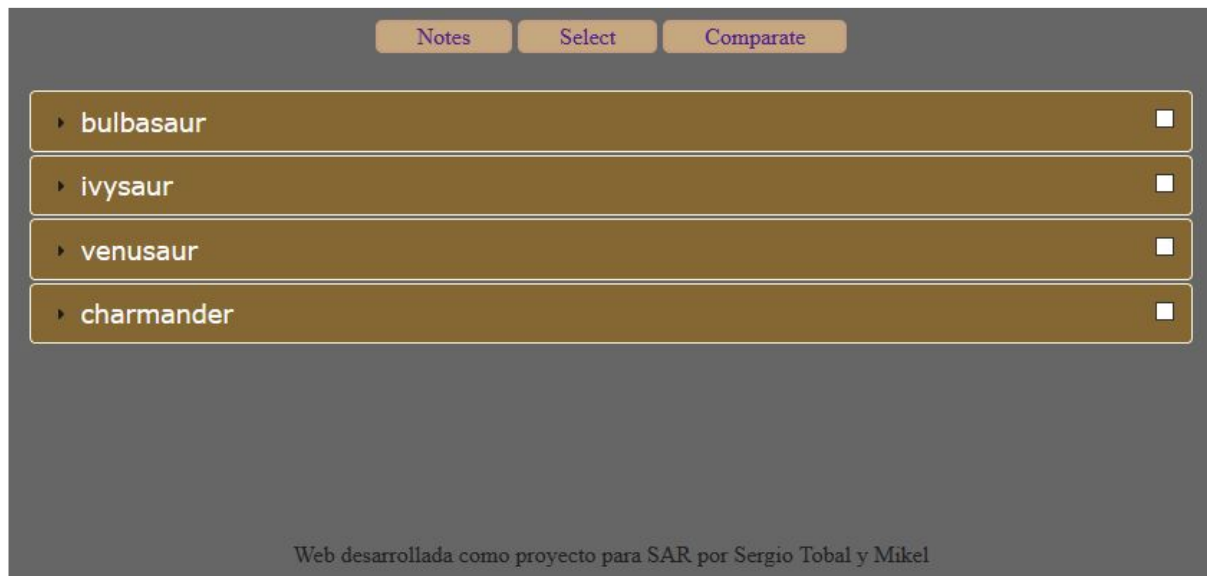
Web desarrollada como proyecto para SAR por Sergio Tobal y Mikel

Fig.3 Vista general de la Página Web (index.php)

En de la parte de comparar tenemos:

- **Select.php:** Aquí es donde hacemos uso de la API, a la cual llamamos para solicitar todos los datos de los Pokemon, tratarlos y mostrarlos en un listado de tipo acordeón. Hemos usado este tipo de listado debido a que hemos querido que al pulsar sobre uno de los elementos para mostrar sus datos, el resto no se muestre. Tras probar varias implementaciones nos hemos quedado con la de *jQuery UI*, ya

que ha sido la que más fácilmente ha funcionado. Una vez mostrados todos los elementos el usuario debe seleccionar 2 elementos a comparar. Es importante tener en cuenta que para realizar la comparación, el sistema toma el primer elemento seleccionado como primer elemento a comparar y como segundo elemento cualquier otro que se elija después. Si se necesita cambiar el primer elemento habrá que recargar la página.

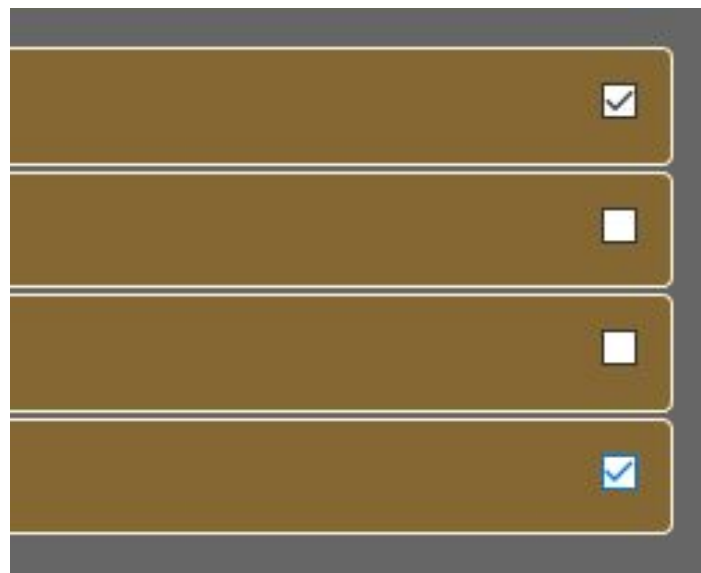


The screenshot shows a web interface with a dark gray background. At the top, there are three buttons: "Notes", "Select", and "Compare", each in a light brown rounded rectangle. Below these buttons is a list of four Pokémon names, each in a brown rounded rectangle with a small white checkbox on the right. The names are "bulbasaur", "ivysaur", "venusaur", and "charmander". At the bottom of the interface, there is a line of text: "Web desarrollada como proyecto para SAR por Sergio Tobal y Mikel".

Pokémon	Seleccionado
bulbasaur	<input type="checkbox"/>
ivysaur	<input type="checkbox"/>
venusaur	<input type="checkbox"/>
charmander	<input type="checkbox"/>

Web desarrollada como proyecto para SAR por Sergio Tobal y Mikel

Fig.4-1 Vista general de la página de selección (select.php)



This screenshot shows the same list of Pokémon as Figure 4-1, but with the first and last items selected. The checkboxes for "bulbasaur" and "charmander" are now checked, while the checkboxes for "ivysaur" and "venusaur" remain unchecked.

Pokémon	Seleccionado
bulbasaur	<input checked="" type="checkbox"/>
ivysaur	<input type="checkbox"/>
venusaur	<input type="checkbox"/>
charmander	<input checked="" type="checkbox"/>

Fig.4-2 Cuando marcamos 2 Pokemons, estamos listos para navegar a la próxima sección

- **Comparator.php:** Al acceder a esta página recuperamos los elementos seleccionados por el usuario en *select.php*, y llamamos a un script de PHP que nos devuelve los valores de esos elementos, con lo que con esto podremos mostrar nuestra gráfica generada por *chartJS*, y con la que mostramos de manera muy clara y visual la comparación entre los 2 elementos.

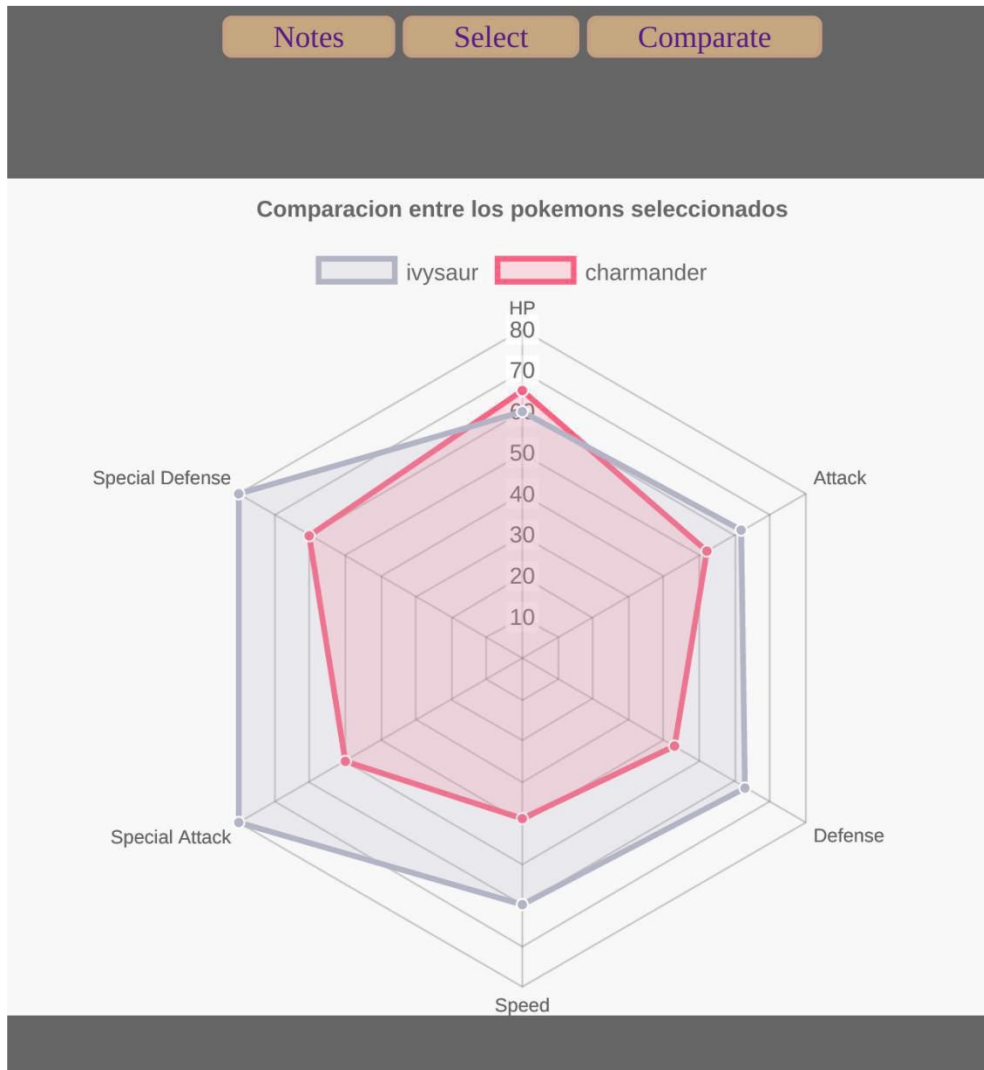


Fig.5 Podemos comparar las estadísticas de los 2 Pokemon seleccionados de manera visual

En *index.php* tenemos 2 funciones que nos permiten guardar información escrita por el usuario para mostrarla más tarde.

- **Contenedor para tomar notas:** Hemos creado, en la página principal, un contenedor donde escribir y tomar notas. Esta función está implementada usando el *localStorage* que fue introducido con HTML5. Lo hemos hecho así porque es todo parte del cliente que no deseamos ni hemos de gestionar. Así que, como todo forma parte del cliente, el usuario escribe en su navegador las notas que necesite, cierra la

web y/o navegador, y cuando vuelva seguirá disponiendo de las mismas notas, siempre que no borre la caché del navegador.

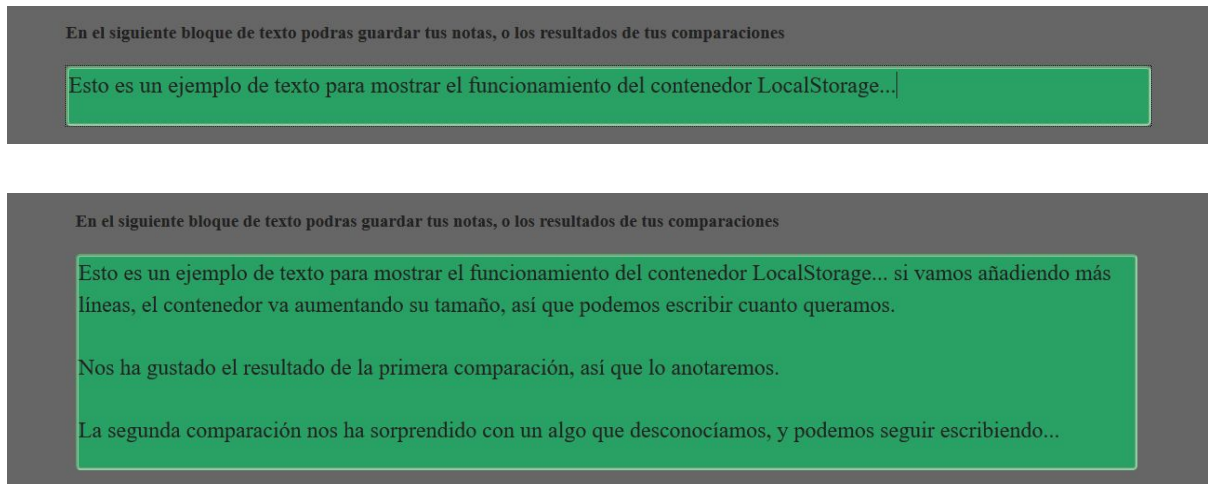


Fig.6 El contenedor para anotaciones en funcionamiento

- **Valoraciones de los usuarios:** En la página inicial hemos pensado que estaría bien que los usuarios pudieran ver lo que opinan otras personas de nuestra web, lo cual nos ayuda a nosotros para poder tener un sitio donde ver qué cosas creen los usuarios que deberíamos mejorar, y de paso creemos que los usuarios se animarán más a usar nuestra web al ver que las valoraciones que recibimos son positivas, y en caso de no ser así nos ocuparemos del problema que nos diga el usuario. Para hacer esto pensamos usar un formulario donde se recogen un nombre como identificador, el correo electrónico y el comentario de la persona, y un contenedor donde se muestra la lista de comentarios. Hemos decidido no mostrar ninguno de los correos de los usuarios en el apartado de comentarios, para evitar que otros usuarios pudieran aprovecharse de esto y llevar a cabo malas prácticas.



Fig.7-1 Obsérvese el caso en el que hay dos comentarios en la sección de valoraciones

Añade tu propio comentario con AJAX:

Nombre:

E-mail:

Comentario:

Comentario sobre la pagina Web.

Fig.7-2 Podemos rellenar el formulario para añadir nuestra valoración

Comentarios acerca de PokeCompare:

nombre (16:28:23, 11 December, 2016)

come1

nomb2 (16:29:23, 11 December, 2016)

com2

Visitante (20:37:20, 12 December, 2016)

Comentario sobre la pagina Web.

Fig.7-3 Y al pulsar sobre el botón “Enviar” nuestro comentario aparece al instante

2.1 Puntos a mejorar

- La API REST a la que llamamos está alojada en un servidor gratuito y es de uso público, así que su tiempo de respuesta no es el mejor. Existe la posibilidad de crear una **base de datos** volcando el contenido de la API pero hemos pensado que se escapa del alcance de lo pensado en cuanto a dificultad y tiempo invertido.
- El bloc de notas o **campo de texto** donde permitimos al usuario guardar la información entre sus visitas sería más conveniente si fuera **flotante** y se mostrase todo el rato, para de esta manera introducir las notas más cómodamente.
- De la parte de selección a la de comparación, pasar como parámetro una **lista**, para así poder **seleccionar infinitos** pokemons y mostrarlos todos en una gráfica en la sección de comparar (habría que ver si conviene limitarlo a algo finito para evitar posibles sobrecargas o una reducción notable en interpretabilidad). Ahora mismo la selección de pokemons coge sólo el primero que ha sido pulsado y cualquier otro

que se pulse después será el segundo pokémon para la comparación.

- Colocar un **indicador de progreso** (*progressbar*) mientras el cliente recibe la información del servidor y cargamos los datos, ya que tal y como está ahora no se sabe si hay algún error en la página o ha ido mal la carga. Es cierto que en el comparador se ha hecho un pequeño control de errores sobre este aspecto y si no se llegan a poder mandar los datos se muestra por pantalla, pero pensamos que es más cómodo y conveniente indicar dichos estados de carga visualmente y en tiempo real.
- Los comentarios que se muestran en la página principal podrían ir dentro de una lista navegable mediante una barra de desplazamiento (*scrollbar*) o podríamos **mostrar sólo 10 comentarios** con una opción de “mostrar más” ya que, si queremos mostrar todos, cuando haya muchos comentarios la web será muy larga, aunque esto también se podría remediar al mostrar los comentarios comenzando por el más reciente y hacia atrás; suponemos que habría menos gente que quisiera mirar los comentarios anteriores.
- Usando CSS3 tenemos una gran cantidad de opciones para hacer que nuestra página sea tanto **visualmente atractiva** como agradable, para que su uso sea mejor experiencia y complementemos un buen funcionamiento con un buen acabado. Sin embargo, para esta práctica nos ha parecido de los aspectos menos importantes y, por tanto, hemos aplicado estilos a ciertos elementos de la página pero no hemos llevado a cabo una mayor profundización.

3. Tecnologías usadas

3.1 HTML5 & CSS3

Aunque se nos ha recomendado el uso de XHTML, hemos decidido usar HTML5 (colocando al inicio de los ficheros `<!DOCTYPE html>`), ya que hemos querido hacer uso de varias características únicas de esta versión como el *sessionStorage*, que nos permite guardar variables en el ámbito de una sesión, haciendo así más sencillo pasar variables de una página a otra.

En el caso de CSS3 lo hemos usado para un par de casos de estilo, aunque al final donde mejor nos ha venido ha sido a la hora de seleccionar todos los elementos que cumplieran unas características, esto lo hacemos con `input[class^='rightHeader'], input[class*='rightHeader']`, que nos sirven para aplicar un estilo a las clases que empiezan o tengan en su nombre *rightHeader*. Ésto nos ha sido muy útil ya que una de las páginas la creamos de manera dinámica y no se sabe exactamente el nombre del atributo *class* hasta que se ejecuta.

Queremos recalcar que todos los valores están definidos mediante proporciones (porcentajes), para que nuestra web tenga un diseño adaptable (*Responsive Web Design*), esto significa que en vez de poner valores fijos, todo sean valores o posiciones relativas, que permitan a los elementos modificar su tamaño dependiendo del tamaño de la ventana de manera fluida.

3.2 JavaScript & jQuery & AJAX

Se nos pedía usar JavaScript, pero al ir estudiando el cómo hacer las cosas hemos ido descartando usos del lenguaje en varios lugares en favor de jQuery.

3.2.1 jQuery Accordion

Por ejemplo, haciendo el listado *acordeón*, la lista de todos los pokemons para seleccionar la creamos al inicio con HTML y JS pero, una vez presentada la lista, sólo se puede hacer clic sobre los textos de cada cabecera para interactuar con la misma. Esto no cumple los estándares de UX (User eXperience), ya que nos gustaría poder interactuar con cada cabecera clicando en cualquier punto del área que comprende la misma. Estudiando otras posibilidades de implementar este tipo de listado hemos llegado a jQuery. Para poder usar jQuery se pide incluirlo en el *header* de las webs que lo usen, y nosotros lo hacemos con

```
<script src = "ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"> </script>
```

para enlazar la librería jQuery proporcionada por Google; de esta manera no malgastamos nuestro espacio en el *hosting* ni el ancho de banda que podría ocasionar el tenerlo en local. Para el acordeón usamos la librería *jQuery UI* que permite implementarlo con la

funcionalidad explicada arriba. Así que en *select.php* tenemos un código parecido al siguiente:

```
$(function() {
    $('#jQuery_accordion').accordion({
        collapsible: true,
        active: false,
        heightStyle: "content",
        activate: function (event, ui) {
        }
    });
    $('#jQuery_accordion input[type="checkbox"]').click(function(e) {
        e.stopPropagation();
    });
});
```

Viendo este código ya se puede suponer que se usa jQuery, debido a la aparición del símbolo \$ dentro de un código JavaScript, y con el método *.accordion* definimos la forma que va a tener el *acordeón* y su funcionamiento.

Con la segunda parte del código, en la que referenciamos a los elementos con identificador de *checkbox*, evitamos que al hacer clic sobre dichos elementos el *acordeón* responda con el pliegue o despliegue de la cabecera donde hemos clicado. Esto lo hacemos porque, al principio, cuando pulsábamos sobre el *checkbox* correspondiente a uno de los Pokémon del listado, se accionaba el despliegue y el *checkbox* se quedaba sin marcar. La solución, como se puede apreciar, es fácil: al usar *stopPropagation()* detenemos el comportamiento de pliegue o despliegue, y por tanto al hacer clic sobre un *checkbox* que esté dentro de una cabecera se actúa sobre el *checkbox* y no sobre la cabecera.

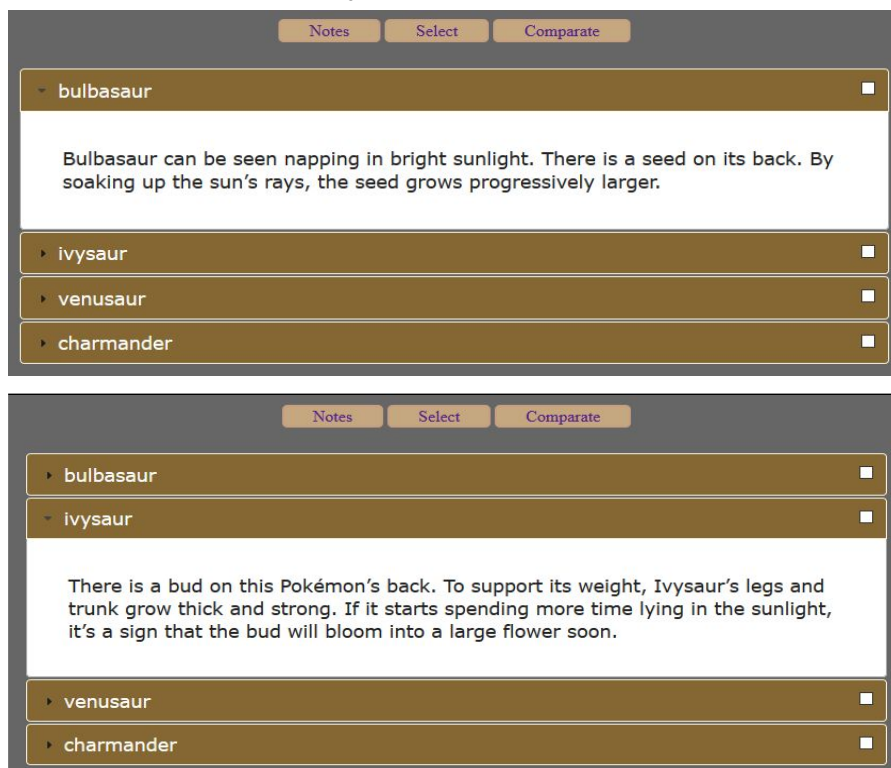


Fig.8 El resultado es un atractivo y, a su vez, conveniente listado

3.2.2 JavaScript SessionStorage

En esta misma página, tenemos otro script que al seleccionar los Pokemons los va almacenando en un objeto llamado *sessionStorage* o *localStorage*. El primero se usa cuando la información guardada se va a usar únicamente en el ámbito de la sesión de la web, y el segundo se usa cuando se quiere almacenar información que se mantendrá aún cuando cerremos la web. Sin entrar muy exhaustivamente a explicar el código, podemos ver que si pulsamos un *checkbox*, cogemos el nombre del Pokemon de esa cabecera, y lo almacenamos en un array asociativo con la key "pokemon1" o "pokemon2".

```
function checkPokemonSelected(obj) {
  if ($(obj).prop('checked')) {
    if (flag == true) {
      sessionStorage.setItem("pokemon1", $(obj).attr('name'));
      flag = false;
    } else {
      sessionStorage.setItem("pokemon2", $(obj).attr('name'));
    }
  } else if (!$$(obj).prop('checked')) {
    if (flag == true) {
      sessionStorage.removeItem("pokemon1");
    } else {
      sessionStorage.removeItem("pokemon2");
    }
  }
}
```

En el apartado de comparación tenemos 2 scripts, uno de ellos es bastante extenso pero muy sencillo de explicar: se trata de la función *mostrarDatos(nombre1, array1, nombre2, array2)* que recibe los nombres de 2 pokemons a comparar, así como los arrays con sus stats, y con esto, después de mirar en qué orden recibimos los stats, los vamos colocando en la gráfica, coloreamos cada área de un color para diferenciarlos y definimos varias opciones de la gráfica generada mediante *chartJS*.

Y la otra función que es *inicializar()*, usada al cargar la página de comparar.

```
function inicializar() {
    var nombrePok1 = sessionStorage.getItem("pokemon1");
    var nombrePok2 = sessionStorage.getItem("pokemon2");

    function ajax() {
        return $.ajax({
            type: 'POST',
            url: 'comparator1.php',
            dataType: "text",
            data: {
                pokemon1: sessionStorage.getItem("pokemon1"),
                pokemon2: sessionStorage.getItem("pokemon2")
            },
            success: function (response, status) {
                response;
            }
        });
    }

    ajax().done(function (result) {
        var stringArray = result.split(/\s+/).filter(Number);

        var array1 = stringArray.slice(0, 6);
        var array2 = stringArray.slice(6, 12);

        mostrarDatos(nombrePok1, array1, nombrePok2, array2);
    }).fail(function () {
        alert("error, no se ha podido llamar a la API");
    });
}
```

Al inicio guardamos los nombres de los 2 pokemons a evaluar (esto lo hacemos por claridad) para luego enviarlos a la función *mostrarDatos*. Se podría no guardar dichos nombres y poner directamente el parámetro de la función en el contenido del *sessionStorage*. Luego, tenemos una función AJAX. La hemos definido así ya que al inicio hemos estado probando con que la función devuelva un valor para trabajar con ese valor en la función padre, pero al final se ha quedado de esta manera porque pensamos que es más claro ver la función por separado. La función en sí es una función de jQuery que realiza una petición HTTP asíncrona (AJAX) para mandar al servidor los nombres de los Pokemons y que la web no se bloquee. También definimos el *dataType* que queremos de vuelta, y mandamos todo esto a la dirección puesta en URL. Veremos en el apartado de PHP qué es lo que hacemos en el servidor y lo que devolvemos.

Cabe mencionar que investigamos otras maneras de hacerlo además de jQuery y AJAX, y vimos para hacerlo con el objeto *SESSION* de PHP, el denominado “método puro” de JavaScript que involucra usar un objeto *XMLHttpRequest*, usar peticiones GET a la siguiente página y mandarlo en la propia URL, pero al final nos decantamos por AJAX, ya que de esta manera no interrumpimos el flujo de la página y el usuario puede seguir

navegando mientras nuestro servidor calcula los datos que se necesitan de vuelta, y los devuelve cuando esté listo.

Por último, podemos mencionar que, cuando la función de AJAX recibe un valor, devuelve otro valor tratado en *ajax().done()*, y se guarda en la variable *result*. Como ya conocemos la estructura de ese valor, que son 12 valores numéricos (6 corresponden a las estadísticas del Pokemon 1, y los 6 restantes al Pokemon 2) separados por espacios, usamos la función *filter* que junto con *split* y una expresión *regex*, separa el contenido de esta variable por espacios, de manera que podemos obtener los valores numéricos e introducirlos en dos arrays. Una vez tengamos esos dos arrays contruidos los enviamos a la función que se encarga de mostrarlos en la gráfica generada por JavaScript, de esta manera será más fácil comparar entre los 6 pares de atributos.

3.3 PHP

Desde su creación, PHP ha sido una de las opciones más elegidas a la hora de crear backends, es un lenguaje robusto y se puede incrustar en las páginas HTML. Aunque su éxito fue bastante alto entre 2004 y 2009, a partir de entonces ha ido considerablemente perdiendo terreno frente a otros lenguajes como Python (*Django*, *Flask*) o Node.js. No obstante, es bastante usado aún, debido a muchas grandes empresas que lo mantienen.

Aunque al inicio se pensó en usar la última versión disponible, PHP 7, debido a algunas características interesantes como el tipado estático o la gran diferencia de rendimiento entre la versión 5 y la 7. Pero la mayoría de hostings gratuitos ofrecían solo la 5, así que al final se ha decidido realizar la *demo* del proyecto con esta versión.

Así que nos instalamos PHP en nuestros ordenadores y con el comando *php -S localhost* arrancamos un servidor con las funciones mínimas para poder probar los cambios que le hacíamos a la web, ya que usar una máquina virtual o subirlo a un *hosting* nos parecía un método ineficiente para probar los pequeños cambios que íbamos realizando.

Otra cosa que también hemos tenido que instalar ha sido *Composer*, el cual es un gestor de dependencias, que se encarga de gestionar librerías con tan sólo introducir el nombre, y posteriormente vincularla a nuestro proyecto con las reglas que le hayamos puesto. Si no dispusiéramos de *Composer*, nos tocaría a nosotros descargar los archivos que componen la librería, integrarlos en nuestro proyecto adecuadamente, y comprobar que no haya ningún fallo, y todo esto tendríamos que hacerlo cada vez que se actualizara el plugin, además de tener que comprobar manualmente de vez en cuando por si el creador de cada plugin hubiera lanzado una nueva versión y actualizarla en nuestro proyecto.

La librería en cuestión que queríamos usar era *pokePHP* y se trata de un *wrapper* PHP de *pokeapi.co*. Permite, en vez de llamar a métodos GET o POST directamente, usar a *pokePHP* como una interfaz que traduce comandos más intuitivos a llamadas GET/POST a la API por nosotros, y posteriormente filtrar la respuesta obtenida por la librería.

Esto es lo que declaramos en la cuarta y quinta línea del siguiente código. En la primera estamos diciendo dónde está el archivo de configuración de *Composer* y en las dos siguientes extraemos los datos que le hemos mandado al servidor mediante una petición POST.

Sabiendo cuáles son los datos que buscamos, hacemos una llamada a la API con los datos y vamos decodificando el archivo JSON devuelto por el *wrapper*, para poder recorrerlo y buscar los datos de cada uno de ellos devolviendolos mediante un *echo* al script que ha mandado las peticiones POST.

```
<?php
require_once __DIR__ . '/vendor/autoload.php';

$id = $_POST['pokemon1'];
$id2 = $_POST['pokemon2'];

use PokePHP\PokeApi;
$api = new PokeApi;
$data = $api->pokemon($id);
$string = json_decode($data,true);
for ($i = 0; $i <= 5; $i++) {
    $number = $string['stats'][$i]['base_stat'];
    echo $number;
    echo "\n";
}

?>
```

3.4 Servicios web - API REST

La definición de *servicios web* dice que se trata de un conjunto de protocolos y estándares que se usan para intercambiar datos entre apps. Antes el protocolo estándar era SOAP, pero se ha ido sustituyendo por REST, ya que mientras que el primero es más estricto y necesita hacer POST a URLs, luego se le tienen que añadir los estándares de SOAP y termina siendo pesado, otros problemas que se le achacan es que la respuesta no es legible directamente por una persona y que su rendimiento es bastante bajo comparado con REST.

Aun así, REST también tiene sus problemas, como que sólo se usa a través de llamadas HTTP o que no es fiable, pero suple sus carencias con creces, y por eso el siguiente gráfico nos muestra cómo desde su creación, REST, no ha parado de ser usado por cada vez más gente.

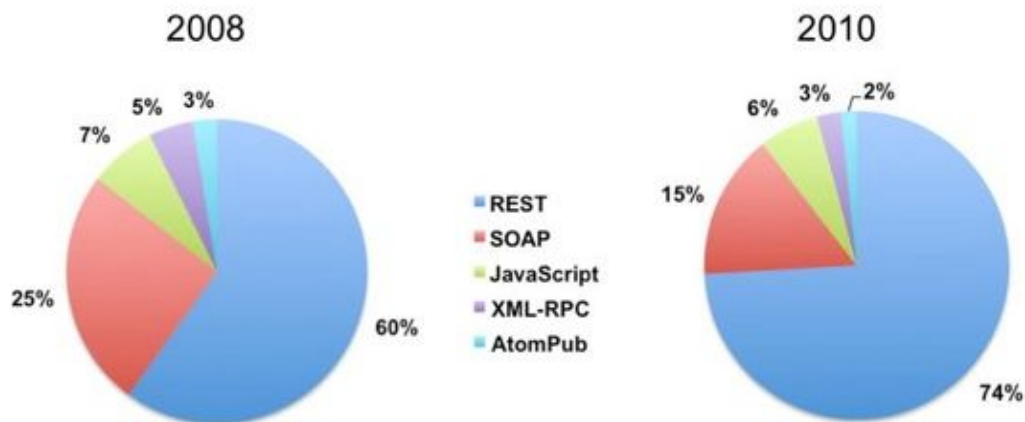


Fig.9 En tan sólo 2 años se observó un notable incremento del uso de REST durante la época en la que tuvo su alza

En este caso hemos usado el ya mencionado *wrapper* de una API REST, así que en vez de hacer peticiones GET a la API de *PokeApi*, usamos el plugin *pokePHP* con el que accedemos a los métodos de la API, nos devuelve el resultado en JSON, lo parseamos y hacemos el tratamiento de datos. Esto es básicamente lo que hacemos en la mayoría de códigos PHP que usamos, varían un poco ya que llamamos a distintos *endpoints* de la API pero, al fin y al cabo, hacen lo mismo: se comunican con la API, piden unos datos, los reciben, y los tratan para mostrárselos al usuario.

```
<?php
require_once __DIR__ . '/vendor/autoload.php';

use PokePHP\PokeApi;
$api = new PokeApi;
$data = $api->pokedex('1');
$string = json_decode($data, true);
?>
```

Las 2 primeras líneas son obligatorias, la primera debido a *Composer*, al que hay que decirle donde está su archivo de configuración, y la segunda es donde decimos el nombre del plugin que vamos a usar.

Las 3 líneas siguientes es la llamada a la API, en la que declarando un nuevo objeto de la *PokeApi* ya tenemos preparada la base para hacer llamadas, y mirando cuales son los endpoints disponibles vamos haciendo las llamadas correspondientes hasta llegar a donde queremos, y mediante la función *json_decode()* conseguimos el JSON con los datos, el parámetro *true* se usa para solicitar la respuesta como un array, esto lo único que cambia es a la hora de recorrer el JSON, que se hace *\$string['pokemon_entries']['name']* en vez de *\$string->pokemon_entries->name*.

3.4 Hosting

Al saber que teníamos que dar una VM con la web operativa o subirla a la nube decidimos subirla a la nube por 2 motivos, porque así nos servía como testeo de fallos y latencia, y porque así no teníamos que andar trabajando en una máquina virtual que siempre ralentiza el trabajo. También nos planteamos usar *Docker* o *Vagrant*, pero lo descartamos debido a que nos suponía un trabajo adicional que no reportaba beneficios visibles. Por tanto, buscamos hosting gratuitos con 2 requisitos: soporte de PHP, y soporte de *Composer*, que es el gestor de dependencias para PHP que necesitábamos para las llamadas a la API.

Al inicio se miraron únicamente *hostings* con soporte a PHP 7, ya que ha mejorado bastante respecto a PHP 5, y preferíamos tener lo último e intentar que las peticiones fueran rápidas. Pero encontrar un servicio de *hosting* (tanto para PHP 5 como para la versión 7) que permitiera acceso SSH para instalar *Composer* y que además fuera gratuito no fue tarea sencilla. Finalmente, decidimos quedarnos con *Heroku* ya que habíamos oído que muchos desarrolladores lo usan para hacer pruebas rápidas, y cumplía todos los requisitos mencionados anteriormente.

Heroku es diferente de los servicios tradicionales de *hosting*, ya que se trata de un PaaS (Platform as a Service). Es decir, ellos te dan una plataforma desde la que decirles cómo quieres el espacio, qué es lo que quieres y de qué modo quieres que te lo proporcionen, y después el Cliente se olvida de la escalabilidad, de la división de recursos, etc. Mientras que en los *hostings* tradicionales te proveen de un entorno muy cerrado (usualmente es un panel con unos botones desde el que se te dice que controles todo), lo cual muchas veces es más de lo que necesitas en temas de gestión, y menos en temas de *plugins*, opciones de software, de bases de datos...

Además, *Heroku* cuenta con una documentación muy completa, y siguiéndola pudimos tener en menos de 2 horas la página web corriendo sin preocuparnos de nada que no fuera el código. La dirección web que se nos proporcionó al crear la web es la siguiente:

<http://polar-savannah-45299.herokuapp.com/>

Cabe comentar que, debido a que la API usada es pública y gratuita, no está pensada para un uso real práctico sino para este tipo de pruebas y ejemplos, así que a veces, al cargar la página de selección de elementos, tarda bastante o nos da directamente error. Si vemos que esto ocurriera varias veces seguidas solamente tendríamos que probar a acceder más tarde.

4. Conclusiones

Ha sido algo más difícil que una práctica normal, ya que hemos pensado primero en la idea y no nos hemos centrado tanto en cumplir sólo lo mínimo pedido, pero creemos que por esto también es algo más realista respecto a lo que nos espera al terminar el grado el año que viene.

5. Referencias

1. <http://www.chartjs.org/docs>
2. <https://devcenter.heroku.com/articles/getting-started-with-php>
3. <https://getcomposer.org/>
4. <http://pokeapi.co/>
5. <https://github.com/danrovito/pokephp>
6. <https://html5.validator.nu/>
7. <http://phpcodechecker.com/>