Ejercicios adicionales transformaciones JSON: API REST Star Wars

Se puede encontrar documentación sobre el API REST en este enlace.

Parte 1: Planetas

Se pretende la programación de las siguientes funcionalidades PHP en una web de pruebas:

1. Lectura del *endpoint* de planetas de la fuente de datos remota (contenido JSON)
2. Procesar los datos y guardar como lista JSON los **nombres (name)** de los planetas (e.g.: [“Tatooine”, “Aldearan”,...]). Guardar el JSON resultante a un fichero de la web de pruebas.
3. Procesar los datos y guardar como lista JSON los **terrenos (terrain)**. Guardar el JSON resultante a un fichero de la web de pruebas.
4. Procesar los datos y guardar como lista JSON los **climas (climate)**. Guardar el JSON resultante a un fichero de la web de pruebas.
5. En el JSON de origen hay un período orbital (orbital_period) y período rotacional (rotation_period). Son, respectivamente, el número de horas de cada año y de cada día.
    a. Por ejemplo, en “Tatooine” rotation_period = 23 y orbital_period = 304
    b. Si se divide orbital_period / rotation_period, se obtiene el número de días en un año. Por ejemplo, 304/23 = 13.217 días por año.
    c. Se pide procesar los datos y guardar como una lista JSON objetos con dos claves:
        i. **name**: <Nombre de planeta>
        ii. **days_per_year** <Resultado del cálculo>
    d. Por ejemplo, [{“name”:”Tatooine”, “days_per_year”:13.217} , { … }]

Parte 2: Personajes

Se pretende la programación de las siguientes funcionalidades PHP en una web de pruebas:

6. Lectura del *endpoint* de personajes de la fuente de datos remota (contenido JSON)
7. Procesar los datos y guardar como lista JSON los **nombres (name)** de los personajes (e.g.: [“Luke Skywalker”, “C-3PO”,...]). Guardar el JSON resultante a un fichero de la web de pruebas.

8. En el JSON de origen hay un “hair_color” (color de pelo).
    a. Se pide procesar los datos y guardar como una lista JSON objetos con dos claves:
        i. **name**: <Nombre del personaje>
        ii. **has_brown_hair:** true | false
        iii. Por ejemplo, [{“name”:”Luke Skywalker”, “has_brown_hair”: false} , { … }]
    b. **has_brown_hair** será true sólo si “hair_color” *contiene* brown. (Ojo con el personaje Owen Lars)
9. En el JSON de origen hay un “films” (películas).
    a. Se pide procesar los datos y guardar como una lista JSON objetos con dos claves:
        i. **name**: <Nombre del personaje>
        ii. **num_films:** <Número de películas>
        iii. Por ejemplo, [{“name”:”Luke Skywalker”, “num_films”: 4} , { … }]
    b. **num_films** será el conteo del número de elementos dentro del array “films”

Parte 3: Naves espaciales

Se pretende la programación de las siguientes funcionalidades PHP en una web de pruebas:

10. Lectura del *endpoint* de naves de la fuente de datos remota (contenido JSON)
11. Procesar los datos y guardar como lista JSON los **nombres (name)** de las naves (e.g.: [“CR90 Corvette”, “Star Destroyer”,...]). Guardar el JSON resultante a un fichero de la web de pruebas.
12. Procesar los datos y guardar como lista JSON los **fabricantes (manufacturer)** de las naves (e.g.: [“Corellian Engineering Corporation”,...]). Guardar el JSON resultante a un fichero de la web de pruebas.
13. En el JSON de origen hay un “consumables” (víveres):
    a. Puede tener valores expresados en días, semanas, meses o años.
    b. Por ejemplo: “consumables”: “1 week” ó “consumables”: “3 years”
    c. Se pide procesar los datos y guardar como una lista JSON objetos con dos claves:
        i. **name**: <Nombre de la nave>
        ii. **consumables_in_hours:** <Víveres convertido a horas>
        iii. Por ejemplo, [{“name”:”CR90 corvette”, “consumables_in_hours”: 8760} , { … }]
    d. consumables_in_hours será el dato “consumables” convertido apropiadamente a horas. Será un número entero.

i. Por ejemplo, “5 days” debería ser 5 * 24 = 120.
ii. Por ejemplo, “2 years” debería ser 2 * 365 * 24 = 17520
iii. Puedes asumir que los años siempre son de 365 días y los meses siempre de 30 días.
iv. Ojo con el plural (1 week vs. 2 week**s**)

Parte 4: Paginación

Los enlaces a *endpoints* de las fuentes de datos mostrados anteriormente siempre devuelven sólo 10 elementos.

Estos endpoints REST admiten **paginación**. Es decir, además de:

https://swapi.dev/api/starships/?format=json

…también podemos solicitar:

https://swapi.dev/api/starships/?page=2&format=json
https://swapi.dev/api/starships/?page=3&format=json
…

Se pide programar en PHP la lógica necesaria para enviar varias peticiones y que cada uno de los ejercicios anteriores se ejecute sobre todas las páginas de datos (no sólo la primera).
Sabrás que no hay más páginas porque la petición de una página inexistente devuelve un código HTTP 404 y el JSON: { “detail”: “Not Found” }

Parte 5: BBDD

Diseñar e implementar una base de datos en la página de prueba que guarde los datos procesados resultantes de los ejercicios anteriores.

En lugar de guardar como ficheros JSON en la web de pruebas, almacenaremos los datos en la base de datos.