JSON y su implementación en C++

Nicolás Bustelo

***Abstract*—El objetivo de este informe es transmitir conocimiento básico sobre Json. Sus múltiples usos, su gran utilidad en el mundo de las APIs y lo más importante, la implementación en el lenguaje de programación C + + gracias a la biblioteca de Niels Lohmann [1]**

***Términos Índice*— JSON, API, Bibliotecas en C++, Formato y notación, Lenguajes de programación, Transferencia de datos.**

I. Introducción

EXISTEN muchos formatos de archivos, y en el intercambio de datos a través de internet, nos enfocaremos en Json o también conocido como JavaScript Object Notation [2]. Es un formato ligero de intercambio de datos que se caracteriza por ser fácil de leer para los humanos y fácil de interpretar para las computadoras. Fue creado en los años 2000 por Douglas Crockford [3], uno de los desarrolladores de JavaScript y hoy en día es ampliamente utilizado en varios lenguajes de programación. Luego en 2006 fue estandarizado por la IETF (Internet Engineering Task Force) y en 2013 por la ECMA (European Computer Manufacterers Association).

Aunque obtuvo la estandarizacion por dos organizaciones muy importantes, lo que potenció la aceptación del formato fueron el crecimiento en gran medida de las APIs, la simplicidad de las estructuras de datos y la creciente popularidad del lenguaje JavaScript. Una de las ventajas que conlleva utilizar JSON en el intercambio de datos es la abstracción del lenguaje, no importa en que lenguaje de programación está escrito el programa, cuando se quieren intercambiar datos se hacen a través de un lenguaje "universal" que evita problemas entre los distintos programas. Así todos deben interpretar un formato y no dependen del lenguaje del emisor y viceversa.

Un formato muy utilizado es XML, el cual Json fue gradualmente reemplazándolo por el lado del cliente. XML fue creado en 1998 y proviene de un lenguaje desarrollado por IBM en 1970 conocido como GML. Cabe notar que XML no es un tipo de formato solamente, sino que es un lenguaje de marcado, un formato para codificar un documento con etiquetas o marcas que contienen información adicional. Por lo tanto, si un proyecto necesitaba documentación o metadata, se prefería XML y si lo que se requería eran datos organizados, se usaba Json ya que como requería menos código, es de menor tamaño y se procesaba más rápido, fue ganando terreno. Si bien no tiene la característica de una validación que tiene XML, como se comentará en la sección III, Json es una mejor opción y hoy en día es el favorito del lado del cliente en la transmisión de datos.

Se constituye de dos partes, por un lado, tenemos a los *objetos* con pares key-value sin orden específico, que se delimitan entre {llaves} y la separación entre los pares se marca con una coma (,). Por otro lado, están las listas de valores ordenados, conocidos en C como *arrays* o arreglos, que se expresan entre [corchetes]. Una característica importante de este formato es el value, puede ser tanto un string, como un número, un bool, un null, un array o un objeto. La separación entre key y value está delimitada por dos puntos (:).

Si el value es un número, siguiendo la precisión de JavaScript, son de doble precisión de punto flotante, son siempre en base diez, pueden tener exponente de diez que se representa con la letra E o e con un signo más o menos indicando si el exponente es positive o negativo y no puede ser infinito. Si el value es un bool puede ser true o false y se expresa sin comillas al igual que si es un null.

Fig. 1: Ejemplo de JSON formateado

Cabe a destacar que a diferencia de XML, Json no tiene comentarios. Si bien inicialmente podía tenerlos, Crockford lo quitó por varios motivos. Consideraba que no eran útiles, dificultaba el proceso de deserialización y la gente estaba abusando de ellos.

La extensión para archivos es ".json" y el IANA o Internet Assigned Numbers Authority es "application/json"

II. Biblioteca Nlohmann

La biblioteca de Niels se puede encontrar en GitHub, es de código abierto y es una de las más usadas para interpretar y armar archivos en formato Json en el lenguaje de programación C++. Está muy optimizada y constantemente tiene actualizaciones por si encuentran algún error. A su vez Lohmann creo una página web [4] para consultar todas las funciones y ver en detalle su armado e implementación.

A continuación, se detallarán algunas de las funciones básicas como para poder tener los recursos y las herramientas necesarias para poder crear, leer o manipular archivos o datos

en formato Json. Muchas funciones son sobrecarga de operadores que simplifican en gran manera la forma de manejar los datos.

En C++ para poder operar json primero hay que instanciarlo, para simplicidad en el código se recomienda utilizar:

```
using json = nlohmann::json;
```

luego de incluir el header *"*`<nlohmann/json.hpp>`*"*.
Para instanciar simplemente se escribe:

```
json name;
```

A. *Parser o Deserializar*

Si lo que tenemos es un string o un archivo, para poder manipular los datos formateados en Json hay que parsear o deserializar. Hay varias formas y muy prácticas de hacerlo:

```
//De un FileSteam
json j4;
file >> j4;

// De un String
json j1 = json::parse("[1,null,false]");
```

Fig. 2: Ejemplo para des-serializar

B. *Serializar*

Una vez que el objeto Json está terminado, podemos guardarlo como string, guardarlo como un archivo o imprimir en consola:

```
//Convertirlo en string
string s1 = j;

//A un FileSteam
file << j.dump() << endl;    //Lo guardamos como string

//Impresión "linda" en consola
cout << setw(4) << j << endl;
```

Fig. 3: Ejemplo de serialización

Note la facilidad de poder utilizar streams tanto en la serialización como en la deserialización.

C. *Lectura*

Una vez instanciado Json podemos, al igual que con clases, operar con funciones miembro para analizar diferentes cosas, por ejemplo, si es un objeto o un array, y además podemos con el uso de la sobrecarga del operador [] buscar el value de una key específica o buscar un elemento dentro de un array:

```
//Algunos ejemplos de is.
j.is_array();
j.is_object();
j.is_number();
j.is_string();
j.is_null();
j.is_boolean();

//Buscamos el valor de una key especifica
string date = j["date"];

//Buscamos un elemento dentro de un array
json tweet = j[1];

//Se puede usar un iterator para mostrar todos los pares key value
for (json::iterator it = j.begin(); it != j.end(); it++)
{
    cout << "Key: " << it.key() << " Value: " << it.value() << endl;
}
// Podemos notar como un dato tipo json al igual que con listas podemos
//conocer el comienzo y el final
```

Fig. 4: Ejemplo de lectura de Json previamente parseado

Algo muy común en la lectura y sobre todo con el operador[] son las *exception*. Usualmente se quiere leer un archivo el cual ya se sabe su estructura o sus keys. En caso de colocar una key no válida o que no está en el Json, devuelve un null, pero si lo estamos asignando a un string o un int, float, etc. el programa lanza la exception 302 ya que es incompatible el null con los tipos de datos mencionados. Si estamos buscando una key dentro de un arreglo o viceversa nos lanza una exception 305 ya que si estamos buscando dentro de un arreglo debemos indicar el orden con un número y si estamos dentro de un objeto debemos indicar el key entre comillas.

D. *Armado:*

Si queremos armar datos en formato Json la biblioteca también lo permite, desde crear un objeto o un array vacío hasta agregar el value correspondiente a cada key:

```
// Creamos un objeto vacio
json j0;
j0["alumnos"] = json::object();

// Cargamos datos de forma manual
j0["alumnos"] = { {"alumno1", 1},{"alumno2", 2} };

// Cargar datos del stream
j2 = "{\"one\": 1, \"two\": 2}";

//De un String literal
json j3 = "[3.14159]"_json;

//Cargamos datos de forma array
json j4 = { 1, 2, 4, 8, 16 };
```

Fig. 5: Ejemplo de armado de JSON

Una sobrecarga interesante a tener en cuenta en el armado es operator+= y la similitud con la función *push_back*.

```
json objeto = { {"one", 1}, {"two", 2} };
objeto.push_back(json::object_t::value_type("three", 3));
objeto += json::object_t::value_type("four", 4);
//object_t::value_type formatea el par key value
//Agrega los pares sin orden especifico

json data;
data = json::object();
// Cargamos pares key value de data a objeto
objeto.push_back(data);
```

Fig. 6: Ejemplo de operador += y función push_back en objetos

Hay que tener en cuenta que cuando se utiliza += o push_back no se asignan de manera ordenada los pares key-value al objeto, por otro lado, si lo hacemos con un array los datos se cargaran de manera ordenada siguiendo el concepto de arreglos.

III. API

Uno de los grandes usos de Json es cómo se mencionó previamente en el intercambio de datos a través de internet y un protagonista muy importante en ese proceso son las API o Application Programming Interface. Es un programa que permite que dos aplicaciones se comuniquen entre sí. Podemos pensarlo como un restaurant, donde tenemos una carta para ver la comida disponible, pedirla a través del mozo y que la preparen en la cocina, en este ejemplo el mozo sería la API, la cocina sería el servidor y la comida que nos traen serían los

datos solicitados. El formato más utilizado en la actualidad para enviar esos datos es Json, ya que son datos ordenados tienen cierta estructura definida. Para poder solicitar datos hay un protocolo y se debe leer la documentación de la API a la cual nos queremos conectar, ya que cada servicio es diferente y tiene sus particularidades como puede ser la url, parámetros específicos o hasta una autorización. Por ejemplo, las aplicaciones como Facebook, Twitter, Uber, el clima entre otras utilizan APIs para poder funcionar en los celulares. En particular el uso más común es para pedir información a un servidor como puede ser el clima de una ciudad, estadísticas de la NBA, cotización de acciones de la bolsa de Nueva York, etc. Para profundizar y buscar una API, se recomienda visitar los siguientes sitios web donde se encuentran más de 10000 APIs públicas:

- https://any-api.com/
- https://www.programmableweb.com/category/all/apis

IV. Ejemplos

En el siguiente link se pueden encontrar ejemplos prácticos de integración de Json con las APIs:

https://github.com/NicoBWin/TT1-JSON

Ambos se solicitan los datos mediante una url y se accede a internet a través del protocolo HTTPS gracias a la biblioteca libcurl [5].

El primero utiliza una API pública para poder obtener el precio del dólar actual y compararlo con el de 04/01/2021. Así mismo se detalla a modo de ejemplo el armado y la lectura de JSON.

El segundo utiliza una API de openweather [6] para poder obtener el clima actual de 4 ciudades precargadas. Notar que a diferencia del ejemplo anterior la API solicita que tengamos una llave o key para poder verificar que es una cuenta valida la que solicita los datos y así poder darnos la información requerida. Para poder visualizar los datos de una manera interactiva se utilizó la biblioteca de allegro5 [7].

Los ejemplos antes mencionados son dos aplicaciones prácticas de la gran cantidad de usos que se le pueden dar. En la vida profesional como ingenieros es de gran utilidad tener un conocimiento claro sobre las comunicaciones a través de internet y el uso de Json, ya que una característica muy importante es que nos independizamos de la plataforma en la que trabajemos, ya sea que necesitamos que un sensor cargue datos a internet, o una máquina nos informe de su estado actual, si se maneja sobre internet es de gran utilidad el formato.

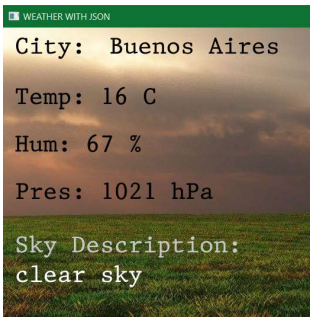

Fig. 6: Ejemplo 2 de API + JSON

V. Conclusion

Como se puede observar a lo largo del informe, leer, modificar y hacer un archivo en formato Json es muy simple con la biblioteca Nlohmann en C++. Asimismo, el potencial de las APIs y la integración con Json hacen del formato un elemento clave en el intercambio de datos. La ventaja de ser un formato claro, sencillo de leer y universal ya que no depende del lenguaje de programación hacen de este, un formato muy versátil y potente.

Claros ejemplos de uso, ya sean personales o profecionales, pueden ser en IoT para cargar datos de los dispositivos a internet, aplicaciones de los dispositivos móviles que utilicen APIs o hasta incluso configuraciones como las que encontramos en Visual Studio.

Referencias

[1] https://github.com/NLOHMANN/JSON
[2] https://www.json.org/json-en.html
[3] https://www.crockford.com/about.html
[4] https://nlohmann.github.io/json/
[5] https://github.com/curl/curl
[6] https://openweathermap.org/api
[7] https://github.com/liballeg/allegro5

Bibliografia

Marrs, Tom (2017). JSON at Work: Practical Data Integration for the Web.

i Code Academy (2017) Json for Beginners: Your Guide to Easily Learn Json In 7 Days.

Nicolás Bustelo: Nació en Buenos Aires, Argentina en el año 1999. Es técnico electrónico y se encuentra estudiando Ingeniería Electrónica en el Instituto Tecnológico de Buenos Aires. Fundraiser en IEEE-ITBA Student-Branch. En el año 2016 obtuvo mención de honor en segundo nivel de la VI Olimpiada Regional de Matemática de la Provincia de Buenos Aires. En el año 2017 obtuvo medalla de oro en dos competencias de las olimpiadas ONIET. Email: nbustelo@itba.edu.ar