**PRACTICAS – MICROSERVICIOS EN C++**

* **1: JSON / XML**: Conversor de ficheros CSV a JSON / XML. Utilizar el fichero
  + Para partir en columnas el fichero CSV:

*void split(const std::string &s, char delim, std::vector<std::string> &elems) {*

*std::stringstream ss;*

*ss.str(s);*

*std::string item;*

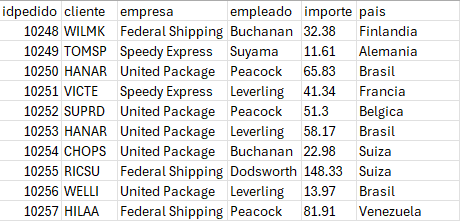
*while (std::getline(ss, item, delim)) {*

*elems.push\_back(item);*

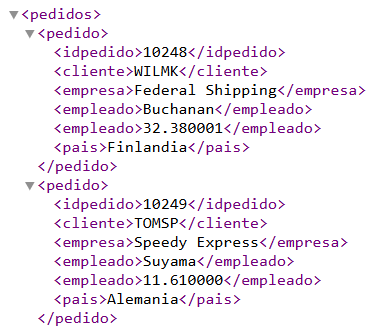
*}*

*}*

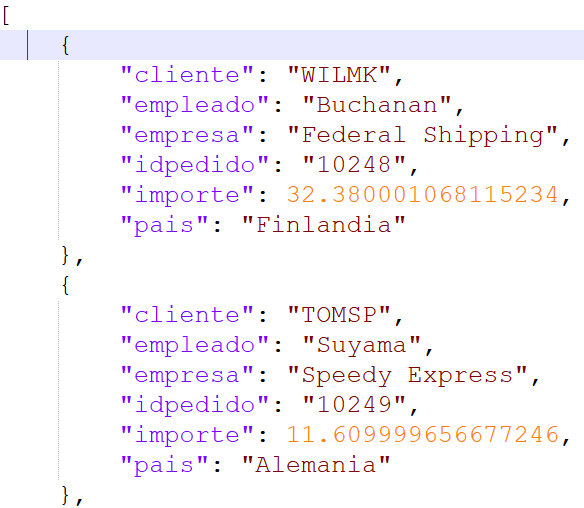
* + Definir una clase con atributos que se correspondan a las columnas del CSV.

**

* + Generar un conversor de pedidos en CSV a XML, grabar el fichero



* + Generar un conversor de pedidos en CSV a JSON, grabar el fichero



* 2: Http: implementar un servidor Http con boost.beast
* 3: Http: implementar un Api Rest para crear, listar y eliminar (solo estas 3 operaciones). ***Activar C++ 17, en propiedades del proyecto 🡪 idioma***
  + Los datos no se almacenan en la BD, los mantendremos en una colección en memoria. Por ejemplo:
    - std::unordered\_map<int, std::string>
  + Las rutas:
    - GET /ítems 🡪 listar los elementos en json
    - POST /ítems 🡪 añade uno nuevo, lo recibe en texto plano o en json
    - DELETE /ítems/{id} 🡪 elimina por ID si existe
    - La colección para almacenar:
    - std::unordered\_map<int, std::string> ítems;
    - int next\_id = 1;
    - #include <boost/beast/core.hpp> // Core de Beast: buffers, streams
    - #include <boost/beast/http.hpp> // Manejo de HTTP: request, response
    - #include <boost/beast/version.hpp> // Versión de Beast
    - #include <boost/asio/ip/tcp.hpp> // TCP con Boost.Asio
    - #include <boost/asio/io\_context.hpp> // Contexto de I/O
* 4: Base de datos: SQLite3 implementar un patrón DAO con las operaciones CRUD y un método select para recuperar una colección de empleados: id, nombre y cargo. En el método créate pasar el empleado por puntero para escribir el id generado en el objeto.