PRACTICAS CURSO DE MICROSERVICIOS EN C++

TEMA 2: SERIALIZACION CON LA LIBRERÍA: nhlomann/json

- **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.

idpedido	cliente	empresa	empleado	importe	pais
10248	WILMK	Federal Shipping	Buchanan	32.38	Finlandia
10249	TOMSP	Speedy Express	Suyama	11.61	Alemania
10250	HANAR	United Package	Peacock	65.83	Brasil
10251	VICTE	Speedy Express	Leverling	41.34	Francia
10252	SUPRD	United Package	Peacock	51.3	Belgica
10253	HANAR	United Package	Leverling	58.17	Brasil
10254	CHOPS	United Package	Buchanan	22.98	Suiza
10255	RICSU	Federal Shipping	Dodsworth	148.33	Suiza
10256	WELLI	United Package	Leverling	13.97	Brasil
10257	HILAA	Federal Shipping	Peacock	81.91	Venezuela

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


```
[
    {
        "cliente": "WILMK",
        "empleado": "Buchanan",
        "empresa": "Federal Shipping",
        "idpedido": "10248",
        "importe": 32.380001068115234,
        "pais": "Finlandia"
    },
    {
        "cliente": "TOMSP",
        "empleado": "Suyama",
        "empresa": "Speedy Express",
        "idpedido": "10249",
        "importe": 11.609999656677246,
        "pais": "Alemania"
    },
```

TEMA 2: crow y boost.beast

- Http: implementar un servidor Http con boost.beast
- 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

- **crow**: implementar un servicio con las operaciones: read, delete, post, put para gestionar los datos almacenados dentro de un unordered_map

  Activar C++ 17, en propiedades del proyecto → idioma

TEMA 3: DOCKER

- Desplegar contenedores de Docker con bases de datos. Utilizar dockerfile con sqlite3, mysql, postgresql y mongo. Restaurar backups en los ficheros dockerfile.

TEMA 3: PERSISTENCIA

- **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.

- Crear un contenedor para **mysql**, y un volumen, restaurar un backup para mysql y luego probar la librería **soci** para conectar con la BD. Cambiar puerto externo para evitar problemas: 3307

- **Microservicio con cache en redis y acceso a bd con mysql**
  - Tenemos dos contenedores: mysql y redis que hay que desplegar y restaurar el backup de mysql con la BD empresa3.
  - El contenedor de redis actuará como cache.
  - Utilizamos la clase EmpleadoRepository para las operaciones CRUD contra los empleados: id, nombre y cargo con la BD de MySQL.
  - La clase RedisCache interactúa con redis con métodos: getEmpleado(id) y setEmpleado(emp) para poder recuperar y escribir un empleado en redis.
  - Una clase EmpleadoService que realiza la siguiente tarea: consulta en redis si el empleado existe lo recupera de redis, si no, tiene que ir a MySQL a recuperarlo.
  - Implementar un servicio en Crow que permita la recuperación de empleados, se apoyara en EmpleadoService.
  - Proyecto en C++ 17
  - Probar el servicio desde PostMan. Conectar al contenedor de Docker (redis) y comprobar si se han creado las claves que hemos solicitado (keys *)

- **CLASES**:

  - Empleado: puede ser una estructura: id: int, nombre: string, cargo:string

  - EmpleadoRepository: La clase que proporciona las operaciones CRUD con la BD de MySQL

- EmpleadoCache: Las operaciones con Redis, para recuperar y grabar.

- EmpleadoService: Capa intermedia, lógica de negocio. Utiliza el repositorio y la caché para comprobar si un empleado esta o no en la cache. Mantiene atributos de la cache y el repositorio.

- MicroServicioCROW: Mantiene un atributo de EmpleadoService, publica una operación GET en /empleado/<int>

- Main: Crear los objetos necesarios: persistencia, servicios, microservicio y lo pone en marcha.

- Crear un contenedor para **postgresql**, y un volumen, restaurar un backup para postgresql y luego probar la librería **libpq** para conectar con la BD. Cambiar puerto para evitar problemas: 5433

- Crear un contenedor para **mongo**, y un volumen, restaurar un backup para mongo y luego probar la librería **mongo-cxx-driver** para conectar con la BD.

TEMA 4: COMUNICACIÓN ENTRE MICROSERVICIOS EN C++

TEMA 5: OPTIMIZACION Y GESTION DE RECURSOS

- Hacer pruebas con **unique_ptr** y **shared_ptr**. Definir una clase Tarea (sencilla, constructor, destructor y ejecutar: muestra un mensaje y se cargan en una colección **queue** añadir punteros de tipo unique y shared, comprobar las copias.

- Implementar una clase Vector utilizando los operadores **new**[] y **delete**[] y comprobar si hay fugas de memoria en el **constructor copia** y el **operator=**

probarlo con **g++** y **valgrind** en **WSL2**. Métodos de Vector: set, add, imprimir hacerlo para el tipo int.

- Convertir la clase anterior, utilizando **unique_ptr**

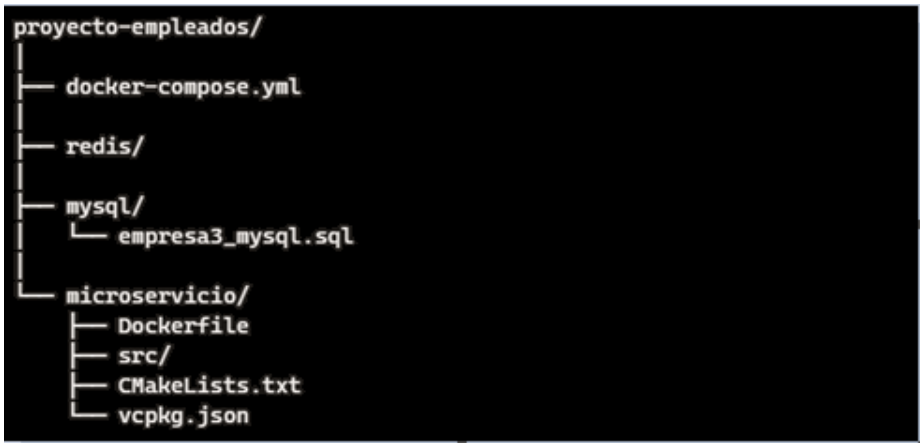
- Implementar una clase **MemCache** que se pueda utilizar en algún microservicio, que haga comprobaciones si el objeto que se pide está previamente en la caché, utilizar una colección **unordered_map**<std::string, T> basada en plantilla. Con los métodos: get / set / clear / remove. Utilizar un **mutex** con **lock_guard**. El método get que devuelva std::**optional**<T> por si acaso no encuentra el elemento en la caché, en ese caso se devuelve: std::**nullopt**. En la llamada habrá que hacer algo como: std::optional<int> resul = cache.get("valor"); if (resul.**has_value**()) { ... } **Aplicarlo al microservicio de crow del tema 3**

TEMA 7 MICROSERVICIOS EN C++ DOCKER Y KUBERNETES

SERVICIO CON CACHE EN DOCKER

Nos apoyamos en la práctica de: "**practica_servicio_con_cache**" hay que escribir un **docker-compose** que lance los 3 servicios necesarios: **redis** y **mysql** como los que teníamos en la otra práctica y ahora hay que añadir un contenedor que despliegue el microservicio. Escribir el **dockerfile** con todo lo necesario, copiando el código fuente, instalando las herramientas necesarias, compilando y arrancando el servicio. Probar desde el navegador a recuperar empleados.

Estructura de carpetas:



El fichero: **vcpkg.json** indica las librerías de vcpkg que necesita el proyecto:

```
{
        "name": "microservicio-empleados",
        "version": "1.0.0",
        "dependencies": [
                {
                        "name": "soci",
                        "features": ["mysql"]
                },
                "hiredis",
                "crow"
        ]
}
```

El código fuente al copiarlo a una imagen de Ubuntu puede tener problemas con LF a final de la línea.

Windows: CRLF

Linux: LF

En Notepad++
Editar → Conversión fin de línea → Fin de línea Linux

El fichero CMakeLists.txt sería algo así:

```
cmake_minimum_required(VERSION 3.15)

project(microservicio-empleados)

# Activar C++17

set(CMAKE_CXX_STANDARD 17)

set(CMAKE_CXX_STANDARD_REQUIRED ON)

# Usar vcpkg como toolchain

# Esto se define en el comando de cmake, este comando va al Dockerfile

#cmake
#DCMAKE_TOOLCHAIN_FILE=/opt/vcpkg/scripts/buildsystems/vcpkg.cmake


# Añadir los ficheros del código fuente:

file(GLOB_RECURSE SOURCES "src/*.cpp")

add_executable(microservicio ${SOURCES})

# Buscar las librerías instaladas por vcpkg

find_package(Crow CONFIG REQUIRED)

find_package(SOCI CONFIG REQUIRED)

find_package(hiredis CONFIG REQUIRED)


# Enlazar las librerías al ejecutable:

target_link_libraries(microservicio PRIVATE

    Crow::Crow

    SOCI::soci_core

    SOCI::soci_mysql

    hiredis::hireredis
```