Introducción a los servicios web en C++

Antonio Espín Herranz

Los servicios web

- Los servicios web permiten que aplicaciones se comuniquen entre sí a través de la web, usando protocolos como HTTP y formatos como XML o JSON. Los más comunes son:

- **RESTful services**: Basados en HTTP, usan métodos como GET, POST, PUT, DELETE.

- **SOAP services**: Más estructurados, usan XML y requieren WSDL para definir la interfaz.

Servicios en C++

- Aunque C++ no es el lenguaje más popular para desarrollo web, tiene ventajas únicas:
  - Rendimiento superior: Ideal para aplicaciones que requieren alta velocidad y bajo consumo de recursos.
  - Control detallado de memoria y recursos: Fundamental en sistemas embebidos o aplicaciones críticas.
  - Interoperabilidad con sistemas existentes: Muchos sistemas legacy están escritos en C++.

Herramientas en C++

- CppREST SDK (Casablanca)
  - Biblioteca moderna para crear clientes y servicios REST
- Boost.Beast / Boost.Asio: Manejar HTTP y TCP
- gSOAP: Servicios Web SOAP en C++
- XmlLite: Analizador XML

Cuando elegir C++

- Máximo rendimiento
  - Video juegos multijugador
  - Comercio financiero

- Integración con sistemas Legacy (sistemas heredados), siguen dentro de la organización a pesar de estar obsoletas.

  - Características de un sistema Legacy:
    - Tecnología obsoleta
    - Difícil de modificar
    - Poca documentación
    - Dependencia crítica: no se pueden retirar
      - Por ejemplo, COBOL

Conceptos básicos de arquitectura web y APIs

Arquitectura de microservicios

- La arquitectura de microservicios consiste en dividir una aplicación en múltiples servicios pequeños, independientes y especializados, cada uno encargado de una funcionalidad específica. Estos servicios se comunican entre sí a través de APIs, generalmente usando HTTP y formatos como JSON o XML2.

- Características clave:
  - Desacoplamiento: Cada microservicio puede desarrollarse, desplegarse y escalarse por separado.
  - Especialización: Cada servicio cumple una función concreta (por ejemplo, autenticación, pagos, notificaciones).
  - Independencia tecnológica: Puedes usar diferentes lenguajes o bases de datos en cada microservicio.
  - Escalabilidad: Puedes escalar solo los servicios que lo necesiten, no toda la aplicación.
  - Resiliencia: Si un servicio falla, los demás pueden seguir funcionando.

Ventajas

•Desarrollo ágil: Equipos pequeños pueden trabajar en paralelo.

•Entrega continua: Puedes actualizar servicios sin afectar al resto.

•Mejor mantenimiento: Más fácil localizar y corregir errores.

•Escalabilidad granular: Solo escalas lo que realmente necesita más recursos.

Diferencias REST / SOAP

Introducción

- El estilo REST *es una forma ligera de crear Servicios* Web.
- Se basan en las URLs.
- Proporcionan acceso a URLs para obtener información o realizar alguna operación.
- Son interesante para utilizar con **peticiones** de tipo **AJAX** y para acceder con **dispositivos con pocos recursos**.

Características

- Sistema cliente / servidor.
- No hay estado → sin sesión.
- Soporta un sistema de caché
- ***Cada recurso tendrá una única dirección de red***.
- Sistema por capas.
- Variedad de formatos:
  - XML, HTML, text plain, JSON, etc.

Recursos

- Un recurso REST es cualquier cosa que sea direccionable a través de la Web.

- Algunos ejemplos de recursos REST son:
  - Una noticia de un periódico
  - La temperatura de Alicante a las 4:00pm

Algunos formatos soportados

Formato	Tipo MIME
Texto plano	`text/plain`
HTML	`text/html`
XML	`application/xml`
JSON	`application/json`

URI

- Una URI, o **Uniform Resource Identifier, en un servicio web RESTful es un** hiper-enlace a un recurso, y es la única forma de intercambiar representaciones entre clientes y servidores.

- Un servicio web RESTful expone un conjunto de recursos que identifican los objetivos de la interacción con sus clientes.

Formato de las peticiones

- La peticiones REST tienen un formato con este:
- http://localhost:8080/app/trabajadores/101
- **trabajadores: representa un recurso.**
- 101:El identificador del Trabajador, es el equivalente a .../trabajadores?id=101
- La URL de REST está orientada a recursos y localiza un recurso.

Verbos REST

- Los verbos nos permiten llevar a cabo acciones con los recursos.

- Se asocian con las operaciones **CRUD.**
  - **GET:** Obtener información sobre un recurso. El recurso queda identificado por su URL. **Operación read**.
  - **POST:** Publica información sobre un recurso. **Operación create.**
  - **PUT:** Incluye información sobre recursos en el Servidor. **Operación update.**
  - **DELETE:** Elimina un recurso en el Servidor. **Operación delete.**

REST vs SOAP

	REST	SOAP
Características	Las operaciones se definen en los mensajes. Una dirección única para cada instancia del proceso. Cada objeto soporta las operaciones estándares definidas. Componentes débilmente acoplados.	Las operaciones son definidas como puertos WSDL. Dirección única para todas las operaciones. Múltiple instancias del proceso comparten la misma operación. Componentes fuertemente acoplados.
Ventajas declaradas	Bajo consumo de recursos. Las instancias del proceso son creadas explícitamente. El cliente no necesita información de enrutamiento a partir de la URI inicial. Los clientes pueden tener una interfaz "listener" (escuchadora) genérica para las notificaciones. Generalmente fácil de construir y adoptar.	Fácil (generalmente) de utilizar. La depuración es posible. Las operaciones complejas pueden ser escondidas detrás de una fachada. Envolver APIs existentes es sencillo Incrementa la privacidad. Herramientas de desarrollo.
Posibles desventajas	Gran número de objetos. Manejar el espacio de nombres (URIs) puede ser engorroso. La descripción sintáctica/semántica muy informal (orientada al usuario). Pocas herramientas de desarrollo.	Los clientes necesitan saber las operaciones y su semántica antes del uso. Los clientes necesitan puertos dedicados para diferentes tipos de notificaciones. Las instancias del proceso son creadas implícitamente.

¿Dónde es útil REST?

- El servicio Web no tiene estado.
- Tanto el productor como el consumidor del servicio conocen el contexto y contenido que va a ser comunicado
- El ancho de banda es importante y necesita ser limitado.
  - REST es particularmente útil en dispositivos con escasos recursos como PDAs o teléfonos móviles
- Los desarrolladores pueden utilizar tecnologías como **AJAX**

¿Dónde es útil SOAP?

- Se establece un contrato formal para la descripción de la interfaz que el servicio ofrece → WSDL.
- La arquitectura necesita manejar procesado asíncrono e invocación.

Protocolos HTTP, HTTPs y WebSockets

HTTP

- Es el protocolo base de la web. Permite la comunicación entre clientes (como navegadores) y servidores mediante un modelo solicitud-respuesta.

- Características:
- Unidireccional: El cliente envía una solicitud, el servidor responde.

- Sin estado: No guarda información entre solicitudes (aunque puede usarse con cookies o sesiones).

- Usa el puerto 80 por defecto.

- Formato textual: Las solicitudes y respuestas son legibles y estructuradas.

HTTPs

- Http secure
  - Es la versión segura de HTTP. Utiliza TLS/SSL para cifrar la comunicación entre cliente y servidor.

- Características:
  - Cifrado de extremo a extremo: Protege datos sensibles como contraseñas o tarjetas.
  - Autenticación: Verifica que estás hablando con el servidor correcto mediante certificados digitales.
  - Integridad: Evita que los datos sean modificados durante la transmisión.
  - Usa el puerto 443 por defecto.

WebSockets

- Es un protocolo de comunicación **bidireccional y persistente** que permite que cliente y servidor intercambien datos en tiempo real sin necesidad de múltiples solicitudes HTTP

- Características:
  - Full-duplex: Ambos lados pueden enviar y recibir datos simultáneamente.
  - Conexión persistente: Se mantiene abierta, ideal para apps en tiempo real.
  - Menor latencia: Los datos se envían tan pronto como están disponibles.
  - Usa el puerto 80 (ws://) o 443 (wss://) dependiendo de si es seguro.

WebSockets

- **Casos de uso:**
  - Chats en vivo
  - Juegos multijugador
  - Notificaciones en tiempo real
  - Streaming de datos

Comparativa

Protocolo	Dirección	Persistencia	Seguridad	Ideal para...
HTTP	Cliente → Servidor	No	No	Páginas web estáticas
HTTPS	Cliente → Servidor	No	Sí	Formularios, login, e-commerce
WebSockets	Bidireccional	Sí	Opcional	Apps en tiempo real

JSON / XML

Parsear datos en Json

- Librería: **nlohmann-json**
- **vcpkg install nlohmann-json**
- **vcpkg integrate install**

- En el proyecto hacemos referencia a:
- **#include <nlohmann/json.hpp>**

Ejemplo

```
#include <iostream>
#include <sstream>
#include <nlohmann/json.hpp>

void testJson() {
   nlohmann::json doc;

   doc["curso"] = "Microservicios en C++";
   doc["horas"] = 25;
   doc["tecnologias"] = { "xml", "json", "rest", "soap" };
   std::cout << "Json: " << doc.dump(4) << std::endl;
   std::cout << "curso: " << doc["curso"] << std::endl;
}
```

Ejemplo 2

```
void strJson() {
   // Se define una cadena Raw: con formato json
   std::istringstream ss(R"({"nombre":"Ana","edad":28,
     "intereses":["programacion","musica","senderismo"]})");
   nlohmann::json doc;

   // Se convierte a json:
   ss >> doc;
   std::cout << "nombre: " << doc["nombre"] << std::endl;
   std::cout << doc << std::endl;
}
```

Vector / Grabar a fichero

- Se pueden definir vectores del tipo: **nlohmann::json** para almacenar objetos json:
  - std::vector<nlohmann::json> array;
  - nlohmann::json grupo;
  - grupo = array; // Se convierte automáticamente
  - **grupo.dump(4)** // Añade indentación, el resultado se puede grabar en un fichero o se imprime por la pantalla.
  - Utilizar operador **<<**, con un objeto **std::ofstream** o **std::cout**

Objetos a Json

```
nlohmann::json Pedido::to_json() const
{
    return nlohmann::json{ {"idpedido", this->idpedido},
    {"cliente", this->cliente},
    {"empresa", this->empresa},
    {"empleado", this->empleado},
    {"importe", this->importe},
    {"pais", this->pais}};
}
```

De JSON a Objeto

Parsear datos en XML

- Librería libxml
- **vcpkg install libxml**
- **vcpkg integrate install**

- En el proyecto hacemos referencia a:

  #include <libxml/tree.h>

Objetos a XML

```
xmlNodePtr Pedido::to_xml() const
{
    xmlNodePtr nodo = xmlNewNode(nullptr, BAD_CAST "pedido");

    xmlNewChild(nodo, nullptr, BAD_CAST "idpedido", BAD_CAST this->idpedido.c_str());
    xmlNewChild(nodo, nullptr, BAD_CAST "cliente", BAD_CAST this->cliente.c_str());
    xmlNewChild(nodo, nullptr, BAD_CAST "empresa", BAD_CAST this->empresa.c_str());
    xmlNewChild(nodo, nullptr, BAD_CAST "empleado", BAD_CAST this->empleado.c_str());
    xmlNewChild(nodo, nullptr, BAD_CAST "empleado", BAD_CAST std::to_string(this->importe).c_str());
    xmlNewChild(nodo, nullptr, BAD_CAST "pais", BAD_CAST this->pais.c_str());

    return nodo;
}
```

Vector / Grabar a fichero

- // Agregar todos los nodos al documento:
- xmlDocPtr doc = xmlNewDoc(BAD_CAST "1.0");
- xmlNodePtr root = xmlNewNode(nullptr, BAD_CAST "pedidos");
- xmlDocSetRootElement(doc, root);

- for (auto nodo : array) {
- xmlAddChild(root, nodo);
- }

- // Guardar el archivo:
- xmlSaveFormatFileEnc(ficheroXML.c_str(), doc, "UTF-8", 1);
- xmlFreeDoc(doc);

De XML a Objeto