# Diseño e implementación de un Servicio Web RESTful para la extracción de información de una base de datos NoSQL para la rehabilitación de miembro superior

S. González Grandía

Universidad CEU San Pablo, Madrid, España, s.gonzalez113@usp.ceu.es

## Resumen

*Las aplicaciones software orientadas a la rehabilitación de pacientes suelen incluir entre sus datos de interés, grandes muestras de datos obtenidas a partir de los diferentes sensores que se utilizan para captar el estado del paciente. Si las cantidades de datos son ingentes, en muchas ocasiones se recurre al uso de bases de datos NoSQL que, por sus características, proporcionan una mayor escalabilidad en cuanto al almacenamiento. No obstante, estas bases de datos tienen que ser integradas, posteriormente, dentro de la aplicación y, la mayor parte de las veces, se requiere la obtención de datos agregados (ya que el uso directo de las lecturas de los sensores, dada su cantidad, puede ser inviable). En este sentido, este proyecto propone el diseño y la implementación de un Servicio Web RESTful orientado a integrar fácilmente este tipo de bases de datos, facilitando así el consumo de datos agregados por parte de la aplicación. Las arquitecturas RESTful, hoy en día, son ampliamente utilizadas como middleware de integración, facilitando la incorporación de sistemas (incluso legados) de una manera rápida, directa y sencilla. Además, debido a su éxito existen multitud de frameworks que permiten la construcción de este tipo de servicios de una manera ágil, utilizando distintos lenguajes de programación. En este proyecto, se ha utilizado Play Framework que permite la construcción de un microservicio RESTful de manera intuitiva y organizada.*

## 1. Introducción

En el proceso de rehabilitación se recogen gran cantidad de datos dado que los sensores obtienen parámetros fisiológicos a una frecuencia muy alta. Para poder ordenar e interpretar estos datos es necesaria la creación de una bases de datos apropiada para el tipo y la cantidad de información recogida en las sesiones de rehabilitación. La mejor solución para este tipo de información son las bases de datos NoSQL, ya que son más flexibles e intuitivas al guardar gran cantidad de datos.

Teniendo en cuenta este contexto, este proyecto es la continuación de otro [1]que afrontó la creación de una base de datos NoSQL para la recogida de datos fisiológicos en distintas sesiones de pacientes diferentes.

Una vez definida esta BBDD el nuevo objetivo es poder integrar esa información en un sistema experto. Sin embargo, dada la ingente cantidad de datos registrados por cada sesión, no es posible una integración directa de todos los datos. Se hace necesario la integración mediante un servicio que, por una parte, oculte la complejidad de las diferentes consultas que se deben realizar para la obtención de los datos y, por otra parte, que ofrezca la posibilidad de reducir la cantidad de datos (ofreciendo la posibilidad calcular la media de los valores de los sensores dentro de una ventana de tiempo definida, por ejemplo). Este proyecto surge ante la necesidad de obtener datos agregados de esta base de datos para su introducción en un sistema experto, por lo que se ha optado por realizar una integración utilizando un Servicio Web.

Los Servicios Web es un tipo de software que permite la invocación de las diferentes funcionalidades de un programa a través de la Web (es decir, utilizando los estándares, protocolos y medios de representación propios de esta red). Por lo tanto, facilitan la construcción de software mediante la reutilización de distintos componentes que se pueden encontrar distribuidos en ordenadores remotos. En la actualidad, se pueden distinguir dos aproximaciones diferentes a la hora de diseñar y construir un Servicio Web.

Por una parte, los Servicios Web tradicionales [2] (basados en la pila de protocolos WS-*), que usan XML para describir las funcionalidades que ofrecen y que utilizan el protocolo SOAP (Simple Access Object Control) de comunicación, han sido la evolución directa de todos aquellos sistemas distribuidos basados en un mecanismo de invocación de métodos. Los estándares asociados a estos servicios dan cobertura a las distintas necesidades de una aplicación empresarial, cubriendo aspectos tan importantes como soporte a transacciones, contratos de servicio, etcétera. Sin embargo, la única conexión real con la Web es la utilización del protocolo HTTP para la transmisión de los mensajes SOAP. Es decir, están enfocados a ser consumidos por máquinas y no están realmente integrados en el ecosistema Web. Generalmente, para construir un cliente software para estos servicios, a partir de la definición de su interfaz (descrita mediante WSDL [3]) se suele obtener un trozo de código que debemos incluir en nuestro software (debiéndolo cambiar cada vez que el servicio varía su definición).

Por otra parte, en la última década, ha surgido una nueva tipología de Servicios Web basados en el estilo arquitectónico REST [4] (el estilo arquitectónico en el que está basada la Web). Estos servicios se integran perfectamente en la Web y eliminan la sobrecarga de definiciones XML, etcétera, que caracteriza a los Servicios Web tradicionales. Por ello, estos servicios están siendo ampliamente utilizados hoy en día. Permiten

una integración más sencilla en cualquier aplicación, a costa de perder características "premium" propias de aplicaciones empresariales (como las transacciones).

Por todo esto, se diseñará un Servicio Web RESTful que servirá de wrapper a esta base de datos, proporcinando una API sencilla y reducida que permita consultar los datos almacenados, con la posibilidad de agregarlos (de momento, calculando medias, pero dejando la posibilidad de ampliar estos cálculos por los que se determinen más adecuados en un futuro).

## 2. Metodología

Para el diseño e implementación, se llevarán a cabo las fases habituales en la construcción de un sistema softwares, resumidas en la figura 1 [5]:

- Análisis: se estudia la base de datos para la que se va a hacer la aplicación y se determinan las entidades que se usarán en las demás fases.
- Diseño: se crean las APIs de RESTful para cada entidad que se reunirán en una clase encargada del tratamiento de datos. Estas APIs serán utilizadas en las consultas.
- Implementación: se implementa todo lo estudiado y diseñado en las fases anteriores con el framework de Play. Se crea la clase controlador que contiene los métodos a los que llaman las consultas HTTP.

A continuación, en las diferentes secciones, se detallará la actividad llevada a cabo en cada una de estas fases.

## 3. Análisis y arquitectura

La base de datos creada en el anterior proyecto estaba planteada para representar datos de pacientes y sus sesiones. Los datos recogidos en las sesiones son IMUs (con una frecuencia de 50Hz) y EMGs (con una frecuencia de 1kHz). Al ser medidos a unas frecuencias tan altas, tenemos una inmensa cantidad de información a tratar y leer por sesión.

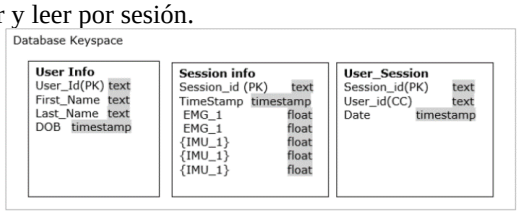


***Figura 1.*** *Tablas de la base de datos*

En esta primera fase, se llevará acabo el análisis de la base de datos que va a ser expuesta mediante el Servicio Web RESTful. Este análisis es necesario puesto que detallará las distintas entidades (y sus relaciones) que deberán ser tenidas en cuenta en posteriores etapas del diseño.

La base de datos (figura 2) almacenaba información relacionada con el paciente (como puede ser el nombre o la fecha de nacimiento). También recogía información de la sesión, almacenando la fecha en la que se realizó y las distintas lecturas que se recogen de los sensores. Como un paciente puede estar asociado a muchas sesiones, también se recoge la relación entre la sesión y el paciente.

En cuanto a la arquitectura del sistema, se ha optado por realizar un Servicio Web RESTful. Este tipo de servicios, se denominan de esta manera puesto que se ajustan al estilo arquitectónico REST (el estilo que subyace en la Web tal cuál la conocemos hoy en día). Es un estilo que define unas restricciones muy sencillas [4]: la comunicación se basa en un protocolo cliente-servidor, todas las entidades importantes de la lógica de negocio son representadas como recursos (que implementan una única interfaz homogénea, en el caso de la Web, el protocolo HTTP) y son expuestas públicamente a través de unos identificadores hipermedia estandarizados (en el caso del Web, las URIs). Además, el intercambio de información entre cliente y servidor se realiza mediante la transferencia de representaciones de los distintos recursos (en la Web, estandarizadas mediante el estándar MIME). Por otra parte, todas las interacciones cliente-servidor deben ser autocontenidas ("stateless", es decir, la respuesta del servidor no debe depender de interacciones pasadas con el cliente). Así, todo el procesamiento que haríamos para ejecutar un proceso en un Servicio Web tradicional (o sistema distribuido tradicional), donde el proceso estaría modelado como una función o método, en un Servicios Web que se ajuste a este estilo arquitectónico, debe modelarse como un conjunto de llamadas HTTP sobre los diferentes recursos del sistema, con el objetivo de modificando su estado (y, en definitiva, el estado de la aplicación).

## 4. Diseño

En [4], Leonard Richardson, analiza cómo se aplican las restricciones arquitectónicas del estilo REST a los Servicios Web (Servicios Web RESTful). Para diseñar una aplicación de este tipo es necesaria la definición de los siguientes elementos: recursos y sus URIs asociadas, API del servicio (métodos HTTP que soportará cada recurso y su semántica asociada) y las representaciones que se intercambiarán entre el cliente y la API.

### 4.1. Recursos

Como se ha comentado anteriormente, los Servicios Web RESTful exponen las entidades más relevantes de su lógica de negocio como si fuesen recursos Web. Por lo tanto, para el diseño de un servicio de este tipo es necesario: definir las distintas entidades centrales del sistema que serán publicadas como recursos, las URIs que tendrán esas entidades (que siendo claras, sencillas, basadas en patrones que permitan su fácil estructuración, ofrezcan una organización jerárquica de dichos recursos), la API de cada uno de estos recursos (esto es, qué métodos HTTP se pueden ejecutar sobre ellos y cuál es su efecto) y el tipo de representación MIME con la que se codificará la información obtenida de cada recurso.

En este apartado, todos los recursos importantes para nuestra lógica de negocio, que han sido descubiertos en la fase de análisis, serán expuestos como recursos Web. Es decir, los pacientes (*patient*) y sus sesiones (*session*) asociadas conforman la parte principal de nuestro sistema. Además, hay que introducir otro tipo de recursos que establecen relaciones de agrupación de elementos. Por ejemplo, el recurso *patients* en plural, hace referencia al

conjunto de pacientes almacenados en la aplicación. De la misma manera se define el recurso *sessions*. Estos recursos son muy importantes en un sistema de este tipo, ya que permiten realizar operaciones de listado de pacientes o de creación de nuevos pacientes (puesto que los métodos HTTP que reciben tienen como objetivo modificar u observar estas colecciones de elementos).

Por otra parte, los recursos *patient* y *session* almacenan los atributos que habíamos identificado en la fase de análisis.

### 4.2. URIs

Una vez definidos los recursos, se les debe asignar una URI. Concretamente, se establecen los distintos patrones que deben seguir las URIs de cada una de las entidades del sistema. Las URIs son jerárquicas, es decir, implican una relación entre los recursos que une. Por ejemplo, la URI correspondiente al recurso *patients* la hemos definido como: */patients* . Mientras que la URI de un paciente (miembro del grupo de pacientes) es la siguiente: */patients/{ID}*, donde {*ID*} representa el identificador que ese recurso concreto presentará en tiempo de ejecución. Y la URI del recurso *session* establecida como: /patients/{*ID*}/sessions/{*ID*}, por un lado establece una relación jerárquica (donde una sesión es una agrupación de sesiones dentro de un paciente que pertenece a un grupo de pacientes), y {*ID*} representa un identificador que tendrá un valor concreto para una *session* específica dentro del sistema.

### 4.3. APIs

Una vez definidos los recursos y las URIs a través de las cuales estarán publicados, es necesario definir qué sucederá cuando se reciba un método HTTP sobre cada tipo de recurso concreto. REST hace especial énfasis en que cada método HTTP se ajuste a la semántica definida en el estándar (por lo tanto, GET se utilizará para obtener información, PUT para actualizar una entidad, POST para crear una entidad nueva y DELETE para borrarla). Así, en base a la definición de la API de todos y cada uno de los recursos definidos, estaremos estableciendo la API completa nuestro Servicio Web.

La tabla 2 (situada en el anexo) muestra la API del recurso *patients*. Con este recurso permitimos la creación de pacientes nuevo (ya que se puede asimilar a añadirlo a la lista existente). También muestra la lista de pacientes existentes mediante el uso de *GET*. Sin embargo, no se permiten los métodos PUT y DELETE puesto que no aplica la actualización de toda la lista ni la eliminación de todos los pacientes a la vez.

La tabla 3 (situada en el anexo) muestra la API para el recurso *patient.* La referencia a este recurso se hace usando su ID. Con un paciente concreto se pueden actualizar sus datos, borrarlos o mostrarlos. Para crear un paciente se hace con sólo la lista, ya que no sé usa el ID porque todavía no tiene uno asignado (no necesita POST).

El recurso *sessions* representa la lista de sesiones de un paciente. En la tabla 4 (situada en el anexo) se puede ver que, igual que en el caso del recurso *patients*, cuando se crea una nueva sesión se hace añadiendo dicha sesión a la lista. No sé puede actualizar ni borrar la lista de sesiones.

| Recurso: Session `/patient/{ID}/sessions/{ID}` | |
|---|---|
| `GET` | Se obtiene una representación JSON de la session con el ID correspondiente. |
| `DELETE` | Se elimina la session con el ID correspondiente. |

***Tabla 1.*** *URIs y APIs para el recurso sesión*

Como muestra la tabla 1, con el recurso *session* se puede mostrar una sesión concreta y eliminarla a partir de su ID. Las sesiones no se pueden actualizar, ya que son medidas tomadas en tiempo real y no pueden cambiarse a posteriori.

### 4.4. Representación

Los Servicios Web RESTful tienen la ventaja de que son muy flexibles. La arquitectura REST y el uso de HTTP permite que los clientes puedan solicitar la representación del recurso de su interés en un formato concreto (XML, JSON, binarios, texto, etc., siempre que el servidor pueda generar dicho formato). En este caso, hemos optado por usar JSON como formato de representación de los datos ya que es fácilmente tratable en cualquier lenguaje de programación.

## 5. Implementación y Pruebas

En esta fase del desarrollo se han utilizado los resultados de la fase de diseño y se ha llevado a cabo la implementación del servicio y su testeo correspondiente.

### 5.1. Implementación

Para la implementación del proyecto usamos Play [7], un framework de desarrollo Web para Java y Scala. Play facilita el diseño de aplicaciones RESTful, lo que lo hace ideal para este proyecto. El código del proyecto está disponible en el repositorio: https://github.com/sofiagrandia/proyectosI.git

Una aplicación en Play se estructura de la siguiente manera:

- Por una parte se define el router. El router relaciona el método HTTP y la URI de cada entidad definida junto con el método que debe atender la petición, es decir traduce una consulta HTTP en una llamada de Java.
- Estas llamadas son métodos definidos en una o más clases denominadas Controladores. Esta clase es vital en Play ya que es la que recibe un input y genera una respuesta utilizando las APIs de REST. El input que recibe es la consulta junto con los recursos, y la respuesta son los datos de la base de datos.
- Las APIs de REST utilizan métodos del paquete *entities*, que también es el encargado de convertir los datos de cada entidad a JSON.

En el siguiente texto, se mostrará como cada uno de los distintos aspectos de PLAY han sido implementados en nuestra aplicación. Para ilustrar las diferentes secciones, se utilizará, como ejemplo, la obtención de una ventana de tiempo para una sesión concreta de un paciente concreto. La petición concreta del paciente en ese caso

sería la siguiente: `GET/patients/id1/sessions/1?init=1612346848743&end=1612346848746`

Este GET parametrizado nos permite realizar consultas más completas, como en este caso, que buscamos las sesiones entre dos instancias de tiempo de un paciente. En la figura 2 se puede ver la URI y el método del nuestro controlador *DBController* (acumula todas las llamadas a la API) que utilizaría esta consulta.

| | | |
|---|---|---|
| GET | /patients/:id/sessions/:idS | controllers.DBController.retrieveSession (id:String,idS:String, init:java.util.Optional[java.lang.Long], end:java.util.Optional[java.lang.Long]) |
| Método HTTP | URI | Método del Controlador |

***Figura 2.*** *Ejemplo de una ruta (Mostrar sesión) en* routes

```
public Result retrieveSession(String patient_id, String id,
java.util.Optional<java.lang.Long> init, java.util.Optional<java.lang.Long>
end) {
   logger.debug("In DBController.retrieveSession(), retrieve session
   with id: {}",id);
        if (DataStore.getSession(patient_id,id) == null) {
           return notFound(ApplicationUtil.createResponse("Session with
           id:" + id + " not found", false));
        }
        JsonNode jsonObjects = null;
        if(init.isPresent()&&end.isPresent()){
            long initial_time=init.get();
            long final_time=end.get();
            System.out.println("Initial_time: "+initial_time);
            System.out.println("End_time: "+final_time);
                    ...
```

***Figura 3.*** *El método* retrieveSession *utilizado en la ruta ejemplo*

```
public static Session getSession(String patient_id, String id){
        Patient patient=getPatient(patient_id);
        return session_patient.get(patient_id).get(id);
    }
```

***Figura 4.*** *Método de* DataStore *para obtener la información de la sesión*

En la figura 4 se puede observar cómo se invoca un método de la clase *DataStore* que proporcionará los datos solicitados. La consulta que se ejecuta en la base de datos se muestra en la figura 5, obteniendo el mismo resultado usando la URI parametrizada mostrada en la figura 6.

```
SELECT * FROM upperlimb.user_sessions WHERE user_id='id1'
muestra las sesiones asociadas al paciente en este caso son las sesiones 0 y 1
SELECT AVG(EMG1) FROM upperlimb.session_info WHERE session_id ='1'
    WHERE timestamp>1612346848743 AND timestamp<1612346848746
muestra la media de los valores de EMG1 entre esas instancias de tiempo
se repetiría para cada valor (EMG2, IMU1...)
```

***Figura 5.*** *Consulta equivalente en la base de datos*

### 5.2. Pruebas

Para hacer pruebas utilizamos POSTman [8], un entorno de desarrollo de APIs que nos permite probar los servicios REST.

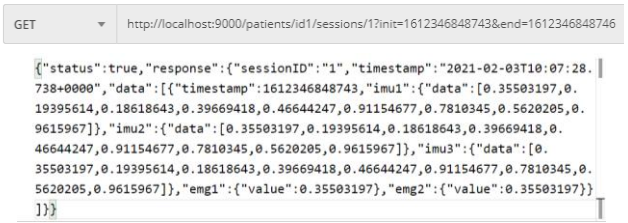


***Figura 6.*** *Resultado de la consulta ejemplo (*http://localhost:9000/patients/id1/sessions/1?init=1612346848743&end=1612346848746*) en POSTman*

Muestra la agrupación de todos los datos de esas sesiones en un único elemento (la media).

### 5.3. Limitaciones de la Propuesta

Inicialmente, la propuesta era permitir la integración de la base de datos NoSQL definida en el proyecto anterior. Sin embargo, esa BBDD se encuentra desplegada en AWS (referencia). Tras infructuosos intentos, no ha sido posible ejecutar la conexión con la base de datos debido a la complejidad técnica del driver y la configuración de seguridad de AWS. Sin embargo, toda la API se ha definido de una manera consistente de forma que sólo queda integrar las llamadas a este elemento.

Para poder comprobar el funcionamiento de la aplicación se han creado un conjunto de datos en memoria que permite la comprobación de las consultas. Estos datos se encuentran en la clase *DataStore*, que contiene todo lo relacionado con el tratamiento de datos, incluidas las listas (HashMaps) de pacientes, sesiones, y una lista que asocia a ambos. Las pruebas del software se han hecho con este conjunto de datos, emulando de esta manera, los datos obtenidos por la base de datos NoSQL

## 6. Conclusiones y Trabajo futuro

El tratamiento de gran cantidad de datos es casi más importante que su almacenaje. Al tener tanta información, para que sea útil, tiene que ser fácilmente legible y representada. Los Servicios Web RESTful facilitan la integración de sistemas diferentes. Aquí hemos conseguido un software que hace de intermediario entre la base de datos y el sistema experto con una aplicación RESTful implementada con el framework de Play. Al haber cubierto todas las fases de esta aplicación, hemos conseguido obtener datos agregados (como puede ser la media) de una manera directa e intuitiva.

Como trabajo futuro quedaría establecer la conexión con la base de datos para poder manipularlos. Como se ha mencionado antes, la limitación de la propuesta es que los datos utilizados están en memoria y por lo tanto, no se ha podido comprobar el funcionamiento de la aplicación con una base de datos.

## 7. Referencias


[1] González Grandía S. Development of a high-performance database for the storage of physiological parameters used for upper limb rehabilitation (2020)

[2] Página Web W3C Working Group Note, Web Services Architecture. Booth D., Haas H. https://www.w3.org/TR/ws-arch/ (Consultada: Enero 2021).

[3] Página Web W3C Working Group Note, Web Services Description Language. Christensen E., Crubera F., Meredith G., Weerawarana S. https://www.w3.org/TR/wsdl.html (Consultada: Enero 2021).

[4] Ricahrdson L & Ruby S. RESTful Web Services. Web Services for the Real World. O'Reilly, 2007 (ISBN: 9780596529260).

[5] Página web Software Developer Central, REST API using Play Framework with Java. (Consultada: Diciembre 2020).

[6] Fielding R.T. Architectural Styles and the Design of Network-based Software Architectures, 2000.

[7] Página Web Play Framework, The High Velocity Web Framework for Java and Scala. https://www.playframework.com/ (Consultada: Enero 2021)

[8] Página Web POSTman, The Collaboration Platform for API development. https://www.postman.com/ (Consultada: Enero 2021)

# Diseño e implementación de un Servicio Web RESTful para la extracción de información de una base de datos NoSQL para la rehabilitación de miembro superior

## Apéndice I: APIs del resto de recursos

| Recurso: Patients `/patients` | |
|---|---|
| `GET` | Se obtiene una representación JSON de la lista de pacientes. |
| `POST` | Se crea un paciente con formato JSON, el ID asignado se usará para obtener la información del paciente más adelante. |

***Tabla 2.*** *URIs y APIs para el recurso pacientes*

| Recurso: Patient `/patients/{ID}` | |
|---|---|
| `GET` | Se obtiene una representación JSON dl paciente cuyo identificador es el ID definido cuando se creó dicho paciente. |
| `PUT` | Se actualizan los datos del paciente con el ID correspondiente. |
| `DELETE` | Se elimina el paciente con el ID correspondiente. |

***Tabla 3.*** *URIs y APIs para el recurso paciente*

| Recurso: Sessions `/patient/{ID}/sessions` | |
|---|---|
| `GET` | Se obtiene una representación JSON de la lista de todas las sesiones del paciente con el ID correspondiente. |
| `POST` | Se crea una session con formato JSON, asignándole un ID que servirá para obtener información sobre la session más adelante. |

***Tabla 4.*** *URIs y APIs del recurso sesiones*