UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE SISTEMAS INFORMÁTICOS

Grado en Ingeniería de Computadores

Proyecto Fin de Grado

Restful: un caso de uso de gestión de bibliotecas

Autor:

Adolfo Rodrigo García Núñez

Tutores:

Isabel Muñoz Fernández
Jorge Enrique Pérez

2016/2017

AGRADECIMIENTOS

En primer lugar, me gustaría agradecer a mi familia todo lo que me han ayudado para llegar hasta aquí. A mi padre, Juan José; a mi madre, María José; a mi chica, Elena; y a mi hermano perruno, Dal. Sin ellos habría sido imposible alcanzar esta meta, sintiendo su apoyo y fuerza en cada bache que me he encontrado por el camino. ¡GRACIAS POR TODO! No puedo olvidar tampoco al resto de mi familia que sin poder estar presentes en todo momento, también he sentido su apoyo.

En segundo lugar, me gustaría agradecer a todos mis amigos por haberme prestado su ayuda cuando la necesitaba y resolver de la mejor manera todos los problemas que me surgían. Sin olvidar todas las experiencias vividas que le hacen a uno mejorar como persona.

Para concluir, me gustaría agradecer a mi tutora Isabel por haberme dado la oportunidad de conocer y profundizar en una nueva tecnología y a mi tutor Jorge por cederme un poco de su gran sabiduría sobre ingeniería del software.

Adolfo Rodrigo

ÍNDICE

Agradecimientos I
Índice III
Índice de figuras VI
Índice de tablas IX
Resumen XI
Abstract XIII
1. Introducción 1
1.1.Web Services 4
1.2. Estado del arte de los Web Services 8
1.3. Objetivos 11
2. Protocolo HTTP 13
3. Estilo arquitectónico REST 29
4. Servicio de biblioteca basado en Web Services 37
4.1. SRS 37
4.1.1. Introducción 37
4.1.1.1. Propósito 37
4.1.1.2. Ámbito 37
4.1.1.3. Definiciones, acrónimos y abreviaturas 37
4.1.1.4. Referencias 37
4.1.1.5. Visión general del documento 37
4.1.2. Descripción general 38
4.1.2.1. Perspectivas del producto 38
4.1.2.2. Funciones del producto 39
4.1.2.3. Características del usuario 39
4.1.2.4. Restricciones 39
4.1.2.5. Suposiciones y dependencias 39
4.1.2.6. Requisitos propuestos 39
4.1.3. Requisitos específicos 39
4.1.3.1. Interfaces externas 39
4.1.3.2. Características del sistema 44
4.1.3.3. Requisitos de rendimiento 51
4.1.3.4. Restricciones de diseño 51
4.1.3.5. Atributos software del diseño 51
4.2. Casos de uso y diagramas de actividades 53
4.2.1. Caso de uso extendido y diagrama de actividad: Registrar usuario 54
4.2.2. Caso de uso extendido y diagrama de actividad: Acceso aplicación 56
4.2.3. Caso de uso extendido y diagrama de actividad: Crear nueva biblioteca 58
4.2.4. Caso de uso extendido y diagrama de actividad: Selección biblioteca 60
4.2.5. Caso de uso extendido y diagrama de actividad: Eliminar biblioteca 62

4.2.6. Caso de uso extendido y diagrama de actividad: Buscar libro 64
4.2.7. Caso de uso extendido y diagrama de actividad: Editar información biblio 66
4.2.8. Caso de uso extendido y diagrama de actividad: Editar información de libro .. 69
4.2.9. Caso de uso extendido y diagrama de actividad: Crear libro 72
4.2.10. Caso de uso extendido y diagrama de actividad: Visualizar todos los libro .. 74
4.2.11. Caso de uso extendido y diagrama de actividad: Cerrar sesión 75
4.2.12. Caso de uso extendido y diagrama de actividad: Eliminar perfil 76
4.2.13. Caso de uso extendido y diagrama de actividad: Eliminar libro 78
4.2.14. Caso de uso extendido y diagrama de actividad: Visualizar/editar perfil 80
4.3. Arquitectura del software 82
4.3.1. Definiciones 82
4.3.2. Estilo arquitectónico seleccionado 83
4.3.3. Descripción arquitectónica 83
4.4. Diagrama de clases 86
4.5. Diagrama de despliegue 87
5. Implementación d alta del servicio 89
5.1. Configuración básica de un servicio REST en java 89
5.2. Securización del servicio: JWT 93
5.3. Diseño y planteamiento del servicio implementado: miBiblioteca 96
5.4 Implementación del cliente web 98
5.5. Balanceo de carga: HAproxy y base de datos MySQL con replicación 99
6. Verificación y validación del servicio. Pruebas 103
6.1. Verificación del servicio REST 103
6.1.1. Login y generación de token de sesión 103
6.1.2. Crear un usuario 104
6.1.3. Eliminar un usuario 104
6.1.4. Obtener información de un usuario buscado por nombre 105
6.1.5. Modificar la información de un usuario 105
6.1.6. Buscar todas las bibliotecas de un usuario 106
6.1.7. Obtener información de una biblioteca 106
6.1.8. Crear una nueva biblioteca 107
6.1.9. Actualizar la información de una biblioteca 107
6.1.10. Eliminar una biblioteca del sistema 108
6.1.11. Obtener todos los libros de una biblioteca 108
6.1.12. Buscar un libro por id y por nombre 109
6.1.13. Crear un libro en una biblioteca 109
6.1.14. Actualizar información de un libro 110
6.1.15. Eliminar un libro de una biblioteca 110
6.1.16. Validez del token recibido 111
6.1.17. Mensaje de error genero por error en el método HTTP enviado 111
6.2. Verificación del cliente web 112
6.2.1. Acceso a la aplicación 112

6.2.2. Creación de un nuevo usuario 112
6.2.3. Selección de bibliotecas 113
6.2.4. Editar biblioteca 114
6.2.5. Menú de inicio de la biblioteca 114
6.2.6. Búsqueda de un libro 115
6.2.7. Formulario crear un nuevo libro 115
6.2.8. Interfaz visualización todos los libros 116
6.2.9. Editar un libro de la biblioteca 116
6.2.10. Editar perfil 117
6.2.11. Salir de la aplicación 117
6.2.12. Caducidad del token 117
6.2.13. Mensaje de confirmación al eliminar un recurso 118
7. Conclusiones y líneas a mejorar 119
Bibliografía XV
Anexos XVII
a. Conversación de Ryan Tomayko con su esposa XVII
b. Código XII
b1. Código ejemplo de un método del servicio REST XII
b2. Código ejemplo de interacción del cliente Web con el servicio XXIV
b3. Código token JWT XXVI
b4. Código tablas MySQL XXVII

ÍNDICE DE FIGURAS

Figura 1. Organización de un sistema distribuido 1
Figura 2. Comunicación cliente con servicio Web 3
Figura 3. Descubrimiento de un servicio Web 4
Figura 4. Descripción de un mensaje WSDL 5
Figura 5. Ejemplo de uso del descubrimiento de servicios (UDDI) 6
Figura 6. Petición-respuesta HTTP 13
Figura 7. Tipo MIME en mensaje HTTP 14
Figura 8. Identificación de un recurso mediante la URL 14
Figura 9. Composición de los mensajes HTTP 15
Figura 10. Flujograma conexión HTTP bajo TCP 17
Figura 11. Ejemplo de proxies en la comunicación 17
Figura 12. Ejemplo uso de caché 18
Figura 13. Ejemplo uso de gateways 18
Figura 14. Túneles-SSL 19
Figura 15. Comunicación cliente-servidor con intermediarios 21
Figura 16. Composición mensaje HTTP 22
Figura 17. Petición-respuesta HTTP 34
Figura 18. Interacción con servicio REST 35
Figura 19. IU de registro 40
Figura 20. IU menú inicio 40
Figura 21. IU búsqueda de un libro 41
Figura 22. IU resultado búsqueda de un libro 41
Figura 23. IU símbolo más en carátula 42
Figura 24. IU tabla todos los libros 42
Figura 25. IU campos formularios 42
Figura 26. Diagrama de casos de uso 53
Figura 27. Diagrama de actividad: Registrar usuario 55
Figura 28. Diagrama de actividad: Acceso aplicación 57
Figura 29. Diagrama de actividad: Crear nueva biblioteca 59
Figura 30. Diagrama de actividad: Selección biblioteca 61
Figura 31. Diagrama de actividad: Eliminar biblioteca 63
Figura 32. Diagrama de actividad: Buscar libro 65
Figura 33. Diagrama de actividad: Editar información biblioteca 68
Figura 34. Diagrama de actividad: Editar información de un libro 71
Figura 35. Diagrama de actividad: Crear libro 73
Figura 36. Diagrama de actividad: Visualizar todos los libros 74
Figura 37. Diagrama de actividad: Cerrar sesión 75
Figura 38. Diagrama de actividad: Eliminar perfil 77
Figura 39. Diagrama de actividad: Eliminar libro 79

Figura 40. Diagrama de actividad: Visualizar/editar perfil 81
Figura 41. Estilo arquitectónico cliente-servidor seleccionado 83
Figura 42. Diagrama de clases 86
Figura 43. Diagrama de despliegue 87
Figura 44. Nuevo proyecto dinámico (eclipse) 90
Figura 45. Resultado primer servicio REST implementado 92
Figura 46. Master status servidor 1 99
Figura 47. Master status servidor 2 100
Figura 48. Prueba login y token 1 103
Figura 49. Prueba login y token 2 103
Figura 50. Prueba creación de usuario 104
Figura 51. Prueba eliminar usuario 104
Figura 52. Estado base de datos usuario eliminado 104
Figura 53. Prueba llamada obtención información de un usuario 105
Figura 54. Prueba llamada edición del perfil 105
Figura 55. Prueba llamada todas las bibliotecas 106
Figura 56. Prueba llamada información de una biblioteca 106
Figura 57. Prueba llamada crear una biblioteca 107
Figura 58. Resultado base de datos creación de biblioteca 107
Figura 59. Prueba llamada editado biblioteca 107
Figura 60. Resultado base de datos editado biblioteca 107
Figura 61. Prueba llamada eliminar biblioteca 108
Figura 62. Resultado base de datos eliminar biblioteca 108
Figura 63. Prueba llamada obtener todos los libros 108
Figura 64. Prueba búsqueda libro por id 109
Figura 65. Prueba búsqueda libro por nombre 109
Figura 66. Prueba respuesta creación libro 109
Figura 67. Prueba base de datos creación libro 109
Figura 68. Prueba respuesta editar libro 110
Figura 69. Prueba base de datos editar libro 110
Figura 70. Prueba respuesta eliminar libro 110
Figura 71. Prueba base de datos eliminar libro 110
Figura 72. Prueba detección token erróneo 111
Figura 73. Prueba respuesta por error en el método 111
Figura 74. Prueba interfaz error acceso 112
Figura 75. Prueba interfaz selección bibliotecas 112
Figura 76. T Prueba interfaz campos formulario 112
Figura 77. Prueba interfaz campo email formulario 112
Figura 78. Prueba interfaz registro correcto 113
Figura 79. Prueba interfaz registro erróneo 113
Figura 80. Prueba interfaz no hay bibliotecas 113
Figura 81. Prueba interfaz muestra de bibliotecas 113

Figura 82. Prueba formulario edición de biblioteca 114
Figura 83. Prueba interfaz selección de bibliotecas con cambios 114
Figura 84. Prueba interfaz inicio 114
Figura 85. Prueba interfaz búsqueda sin resultados 115
Figura 86. Prueba interfaz resultado búsqueda 115
Figura 87. Prueba información libro encontrado 115
Figura 88. Prueba interfaz campos del formulario crear libro 115
Figura 89. Prueba mensaje creación libro 115
Figura 90. Prueba tabla todos los libros 116
Figura 91. Prueba búsqueda en tabla todos 116
Figura 92. Prueba formulario de edición de libros 116
Figura 93. Prueba formulario del perfil 117
Figura 94. Prueba mensaje del cliente por caducidad del token de sesión 117
Figura 95. Prueba mensaje de confirmación 118

ÍNDICE DE TABLAS

Tabla 1. Comparativa básica REST-SOAP 9
Tabla 2. Descripción mensajes HTTP 10x y 20x 24
Tabla 3. Descripción mensajes HTTP 30x 25
Tabla 4. Descripción mensajes HTTP 40x 25
Tabla 5. Descripción mensajes HTTP 50x 25
Tabla 6. Descripción cabeceras HTTP 1 26
Tabla 7. Descripción cabeceras HTTP 2 27
Tabla 8. Caso de uso extendido: Registrar usuario 55
Tabla 9. Caso de uso extendido: Acceso aplicación 56
Tabla 10. Caso de uso extendido: Crear nueva biblioteca 59
Tabla 11. Caso de uso extendido: Selección biblioteca 60
Tabla 12. Caso de uso extendido: Eliminar biblioteca 62
Tabla 13. Caso de uso extendido: Buscar libro 65
Tabla 14. Caso de uso extendido: Editar información biblioteca 67
Tabla 15. Caso de uso extendido: Editar información de un libro 70
Tabla 16. Caso de uso extendido: Crear libro 72
Tabla 17. Caso de uso extendido: Visualizar todos los libros 74
Tabla 18. Caso de uso extendido: Cerrar sesión 75
Tabla 19. Caso de uso extendido: Eliminar perfil 76
Tabla 20. Caso de uso extendido: Eliminar libro 78
Tabla 21. Caso de uso extendido: Visualizar/editar perfil 81
Tabla 22. Anotaciones JAX-RS 89

RESUMEN

El apasionante mundo de las tecnologías Web no para de crecer a pasos agigantados, debido en buena parte, a los avances tecnológicos que hay. Cada vez con más frecuencia, en la mayoría de actividades que realizamos utilizamos como intermediario una aplicación web o una aplicación móvil. Debido a esto, surgen nuevas tecnologías y métodos de implementar estos sistemas y su forma de interactuar con la información. Con el presente proyecto realizaremos un servicio web basado en REST, tecnología en pleno auge, que ofrecerá las funciones básicas para gestionar bibliotecas personales a cualquier tipo de plataforma, por lo que a su vez, se implementará un cliente web que consuma todas las funciones que oferta el servicio.

Las funciones básicas del servicio implementado cubren la posibilidad de crear diferentes tipos de bibliotecas e incluir en ella cualquier libro que se posea, así como la búsqueda de cualquiera de ellos y obtener su posición en la librería.

Antes de desarrollar el proyecto, se ha realizado un estudio de los actuales sistemas distribuidos y Web Services disponibles para así conocer por qué usamos una tecnología u otra. Una vez decidido, se avanzará para entrar a explicar qué es REST y su protocolo de comunicación estrella, HTTP.

Tras analizar de manera teórica los conceptos previos necesarios para realizar este proyecto, se realizará un estudio de ingeniería del software para modelar el sistema implementado. Una vez hecho esto, se procederá a realizar la implementación del servicio web y el cliente web.

Para complementar el proyecto realizado, se implementará un balanceador de carga para la conexión a las dos bases de datos desarrolladas las cuales, se encuentran en un estado de replicación master-master.

ABSTRACT

The exciting web technologies's world is growing by leaps and bounds due to the technological advances that appears. More and more frequently, we usually use a web application or a mobile app as an intermediate in most of the activities we do. As a result, new technologies and ways to use them have arisen. In this project we will perform a Web Service based on REST, a full swing technology, which will offer basic functions to manage personal libraries for a different sort of systems. because of that, it will be developed a web client who uses all those functions.

The basic functions of the developed service let you create different kind of libraries and include any book you have in it as well as all the searches of it and figure out where the book is in the library.

Before developing the project, we will do a study of the current distributed systems and the different Web Services available for knowing which one we will use. Once decided we keep going explaining what REST is and its most important known protocol, HTTP.

After analyzing theoretically the previous concepts we needed to make this project, we will do a software engineer's study to shape the developed system. Once it has been done, we will continue the Web Service and Web Client have been created.

To complement the project, a load balancer will be implemented for the connection to the two databases developed, which are in a state of replication

1- INTRODUCCIÓN

En 1945, momento en el que podemos considerar el inicio de la era moderna del ordenador, empezaron a aparecer los primeros ordenadores. Estos eran realmente grandes y muy caros. Estos ordenadores eran denominados *mainframes* u ordenadores centrales. Por esta época, Herb Grosch acertó asegurando que el poder de procesamiento de una CPU (*Central Processing Unit*) es proporcional al cuadrado de su precio, es decir, cuanto más pagáramos por una CPU, más poder de procesamiento tendríamos. Este concepto ha sido válido hasta el año 1985, momento en el que aparecieron los microprocesadores que, en poco tiempo avanzaron de manera exponencial llegando a alcanzar CPUs de 64 bits, siendo estas mucho más potentes que los *mainframes* y teniendo un precio muchísimo más reducido.

A su vez, otros avances en esta época propiciaron que los ordenadores centrales empezaran a desaparecer. Estos avances fueron las nuevas redes de ordenadores de alta velocidad, las LAN (*Local-Area Network*) permitiendo la conexión de varios ordenadores dentro de un mismo edificio, y las WAN (*Wide-Area Network*) permitiendo una conexión entre máquinas de todo el mundo.

La posibilidad de poder utilizar varios ordenadores igual o más potentes que los antiguos *mainframe* a un precio reducido junto con el avance de las redes de ordenadores que permitían una conexión entre los mismos, dio lugar a la creación de lo que hoy en día conocemos como sistemas distribuidos. En la figura 1 se puede visualizar la estructura de un sistema distribuido.

Existen muchas definiciones que explican qué es un sistema distribuido, pero algunas se contradicen entre sí o no dan una visión clara de lo que es. Por ello, nos centraremos en dar una definición más sencilla pero que a su vez abarque el concepto principal. Por esto, definimos un sistema distribuido como un conjunto de ordenadores independientes que aparecen ante el usuario del sistema como un único ordenador.
Esta definición engloba tanto el hardware como el software, creyendo en ambos casos que hablamos de un único ordenador. Pero en realidad el resultado final, viene dado por un conjunto de ellos.

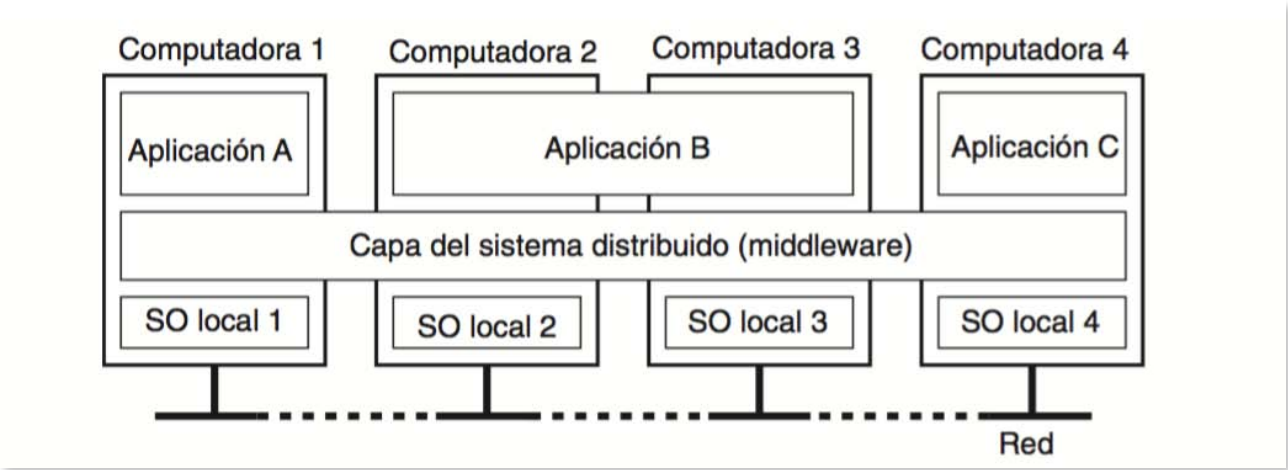


Figura 1. Organización de un sistema distribuido

A raíz de los avances comentados anteriormente, surgen otras técnicas de comunicación entre máquinas, las RPC (*Remote Procedure Call*), permitiendo que un ordenador ejecute código en otro de forma remota sin preocuparse de la comunicación entre ellas.

Como consecuencia de las RPC, salen a la luz otros modelos con características mejoradas. COM (*Component Object Model*), creado por Microsoft y CORBA (*Common Object Request Broker Architecture*), desarrollado por el Object Management Group. El problema de estos modelos era la falta de interoperabilidad que existía entre ellos debido a que las dos máquinas involucradas en la comunicación debían soportar el mismo modelo, bien COM, o bien, CORBA.

Posteriormente, Microsoft mejoró COM y pasó a llamarse DCOM (*Distributed Component Object Model*). Al mismo tiempo, Sun sacaba RMI (*Remote Method Invocation*). Estos nuevos modelos seguían teniendo el mismo problema de interoperabilidad entre máquinas. En este caso debido a que RMI dependía del lenguaje de programación Java.

Fue tras estos avances cuando Microsoft, en 1997, empezó a interesarse acerca de la computación distribuida basada en XML (*eXtensible Markup Language*). La idea que tuvo Microsoft para mejorar la interoperabilidad era permitir que las aplicaciones se conectaran mediante RPCs, utilizando dos estándares que hoy en día son muy conocidos: XML y HTTP (*Hypertext Transfer Protocol*).

Entonces, en 1998, Dave Winer desarrolló XML-RPC, que es un protocolo de llamadas a procedimientos remotos basado en XML. A diferencia de otros protocolos RPC conocidos, XML-RPC era muy simple ya que definían pocos tipos de datos y comandos.

Microsoft cooperó en el desarrollo de XML-RPC pero, considerando que era muy simple, empezó a añadir más tipos de funciones, creando de esta manera SOAP (*Simple Object Access Protocol*). La diferencia más notable entre SOAP y XML-RPC se encontraba en que con SOAP, cada parámetro tenia un nombre determinado y no importaba el orden.

Fue ya en 2003 cuando una nueva versión de SOAP fue recomendada por el W3C (*World Wide Web Consortium*). En los siguientes años, SOAP fue mejorando y empezó a formar parte de una capa subyacente más compleja sirviendo de modelo de desarrollo de *Web Services*, tecnología que permite intercambiar datos entre aplicaciones, introduciendo conceptos como WSDL y UDDI, que analizaremos más adelante.

Mientras se desarrollaba SOAP, en 2002 aparecía REST, intentado revolucionar la forma en la que se comunicaban las máquinas entre sí, sin desarrollar nuevos protocolos o estándares sino aprovechando directamente el propio protocolo HTTP y el lenguaje de marcado XML, intentando imitar el funcionamiento de la Web de entonces.
Pero no será hasta 2009 aproximadamente cuando empezará a tener más relevancia.

Tras ver la evolución de los sistemas distribuidos y recalar en los sistemas mas actuales como son los *Web Services*, conviene detallar los cuatro componentes básicos de las aplicaciones distribuidas que siguen un modelo cliente-servidor, que es el que nos interesa en este proyecto:

- **Servidor**

  Proceso que se ejecuta en un computador que está conectado a una red. Está a la escucha en un puerto, esperando las peticiones de los clientes. Por ejemplo, un servidor Web escucha en el puerto 80. Un computador que ejecuta un servidor de aplicación necesita estar conectado a la red para responder a las peticiones de los clientes.

- **Cliente**

  Programa que ejecuta el usuario de la aplicación. El cliente hace sus peticiones al servidor a través de la red. Por ejemplo, un navegador Web.

- **Protocolo de aplicación para la comunicación entre el cliente y el servidor**

  El protocolo define el tipo de mensajes intercambiados. Por ejemplo, el protocolo de la capa de aplicación de la Web, HTTP, define el formato y la secuencia de los mensajes transmitidos entre el navegador y el servidor Web.

- **Formato de los mensajes que se intercambian**

  Elección del formato de mensaje para la comunicación entre ordenadores. Como por ejemplo, un mensaje XML

Todos estos componentes interactúan entre sí en cada conexión. En la figura 2 se puede observar dicha relación.

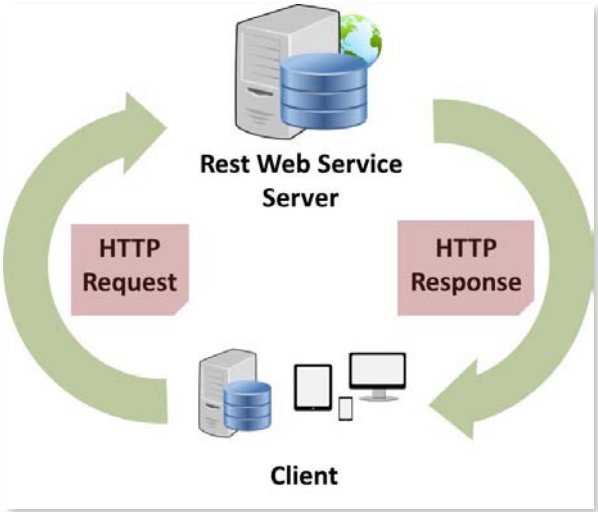


Figura 2. Comunicación cliente con servicio Web

1.1. Web Services

Definir bien un sistema distribuido no es sencillo debido a las diferentes acepciones y características existentes. Al tratar de explicar qué es un *Web Service*, ocurre lo mismo. Existen muchas definiciones sobre ellos, lo que demuestra lo complejo que es todo lo que engloban e implican.

Pero podemos definir un *Web Service* como un conjunto de aplicaciones o de tecnologías con capacidad para interoperar en la Web. Estas aplicaciones o tecnologías intercambian datos entre sí con el objetivo de ofrecer unos servicios. Los proveedores ofrecen sus servicios como procedimientos remotos y los usuarios solicitan un servicio llamando a estos procedimientos a través de la Web.

Un *Web Service* no deja de ser un medio por el cual una serie de máquinas conectadas en red pueden comunicarse y operar entre ellas, usualmente esta conexión consta de peticiones y respuestas entre un cliente y un servidor.

El paradigma de los *Web Services* permite que aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma interactúen en modo cliente/servidor sobre redes de tipo Internet. Las aplicaciones servidoras ofrecen servicios Web accesibles a las aplicaciones cliente a través de Internet. Para conseguir esta comunicación entre máquinas se utiliza una serie de protocolos y estándares, que veremos más adelante.

La comunicación entre máquinas mediante *Web Services*, como se observa en la figura 3, proporcionan:

- Presentación de información dinámica al usuario
- Interoperabilidad. Solventan de esta forma uno de los mayores problemas que surgieron con los primeros modelos de sistemas distribuidos
- Extensibilidad entre aplicaciones

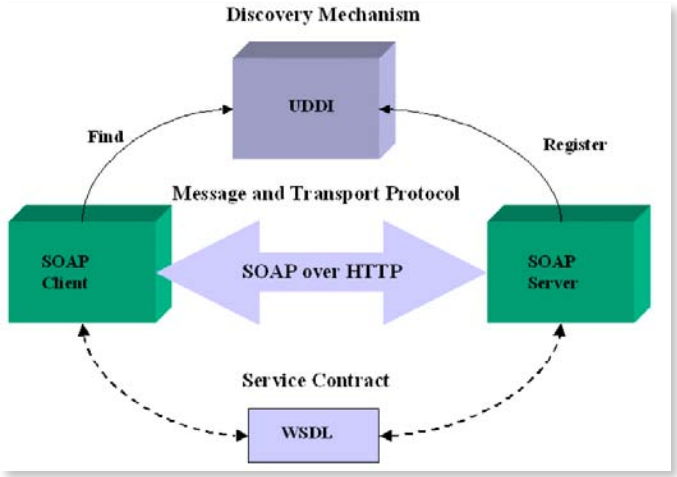


Figura 3. Descubrimiento de un servicio Web

A continuación, vamos a hablar de los protocolos y estándares más conocidos que hacen que los *Web Services* sean tan versátiles. Todos estos protocolos y estándares están recogidos en lo que se denomina la pila de protocolos para Servicios Web.

Con estos protocolos se consigue definir, localizar, implementar y hacer que un Servicio Web interactúe con otro.

Esta pila de protocolos la podemos dividir en cuatro áreas diferentes:

- **Protocolo de comunicación**

  Protocolos de la capa de aplicación y estándares responsables del transporte de mensajes entre las aplicaciones de red tales como HTTP, SMTP, FTP. Algún otro más reciente puede ser BBB (*Blocks Extensible Exchange Protocol*) .

- **Mensajería**:

  En este apartado se agrupan todos aquellos protocolos responsables de la codificación de mensajes en un formato común ya sea XML, JSON o cualquier otro que pueda ser transmitido. La codificación de estos mensajes permiten que sean fácilmente entendidos tanto por el cliente como por el servidor. Actualmente, diferentes tipos de web services como XML-RPC, SOAP o REST, utilizan alguno de estos mensajes en la comunicación entre máquinas.

- **Descripción del Servicio**

  Esta descripción es una interfaz pública para todos los servicios web. Basado en XML-schema, describe la forma de comunicación matizando parámetros como el protocolo, formato de los mensajes transmitidos, etc.
  El cliente, al conectarse a un servicio Web, puede leer esta descripción y ver las funciones disponibles y en el caso de SOAP, interactuar con las mismas.
  La estructura de este fichero de descripción se detalla en la figura 4:

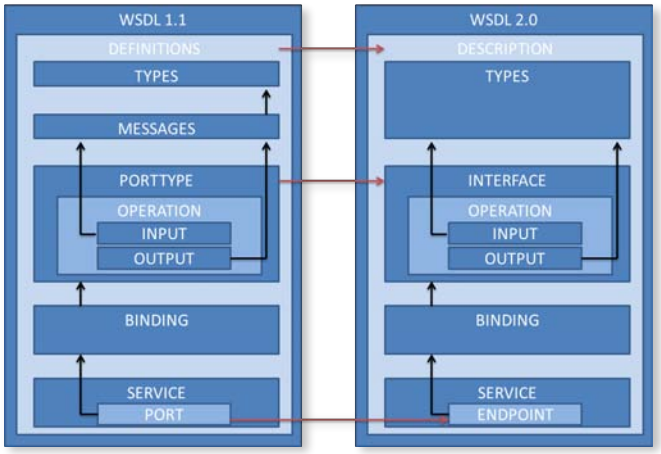


Figura 4. Descripción de un mensaje WSDL

Descubrimiento de servicios

Estándar básico de los servicios web para poder registrar y localizar cualquier servicio Web creado. Se puede considerar como un directorio de todos los servicios Web donde se especifica información acerca de estos como la capacidad, ubicación, requerimientos, etc.
Esto hace que sea fácil descubrir qué servicios están disponibles en la red. Actualmente, la API Universal Description Discovery and Integration - UDDI se utiliza, por lo general, para el descubrimiento de servicios.
En la figura 5 se detallan los pasos necesarios para poder consultar el servicio en este registro.

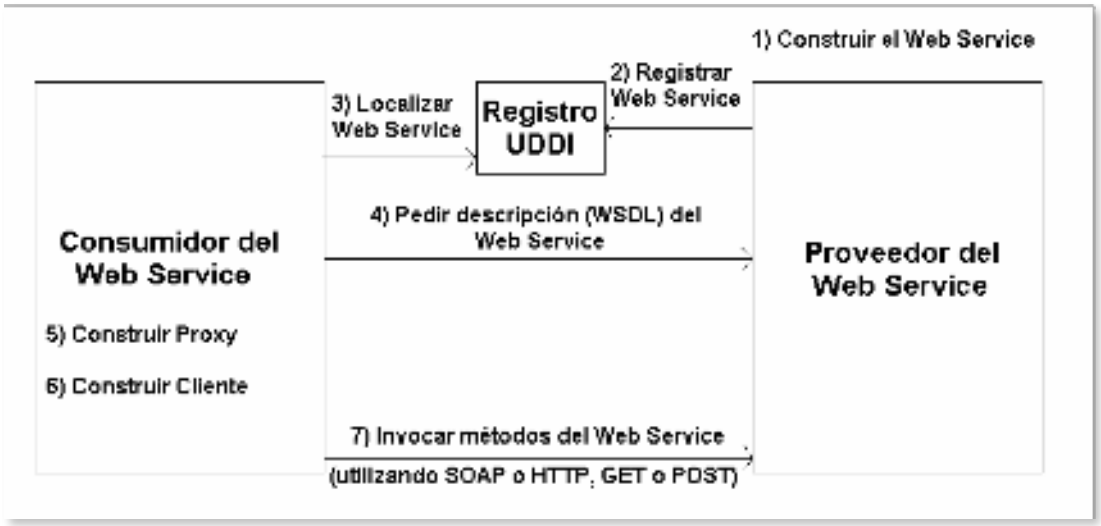


Figura 5. Ejemplo de uso del descubrimiento de servicios (UDDI)

La web hace uso de protocolos y estándares que hoy en día consideramos universales y la mayoría de aplicaciones hacen uso de ellos. Hablamos de HTML y HTTP. Por esta razón, es tan fácil acceder a los recursos en red desde cualquier sitio.

Con los *Web Services* ocurre algo parecido. Estos están evolucionando a pasos agigantados y empresas como Microsoft e IBM intentan hacer que los *Web Services* sean igual de fáciles a nivel de accesibilidad como lo es el servicio web (WWW), haciendo uso de estos protocolos.

Para que exista una evolución de estas tecnologías, los protocolos y estándares comentados se están redefiniendo constantemente. De todos modos, con el paso del tiempo van saliendo otras alternativas, aunque algo incompletas todavía como para poder implementarlas. Algunas de estas alternativas son las siguientes:

En el caso de HTTP y SOAP/REST otras opciones pueden ser :

- Jabber, protocolo asíncrono de transporte (más ligero que HTTP) o EbXML que está pensado para integración de servicios en soluciones B2B (*Business to Business*)
- HTTP-POST, que se basa en XML-RPC

Otras opciones que sustituyen al WSDL son:

- RDF (*Resource Description Framework*), definido por el W3C. Es más potente pero también más complejo que WSDL
- DAML (*DARPA Agent Markup Language*), definido por la agencia de defensa estadounidense. Es también más potente pero más complejo que WSDL

Una propuesta alternativa realizada por Microsoft e IBM, llamada WS-Inspection Language, podría ser un avance del UDDI.

A continuación hablaremos, sobre los modelos más utilizados hasta la fecha para implementar *Web Services*.

En 1998, surge la primera aparición de los *Web Services* en el mundo de Internet, de la mano de SOAP, protocolo que define cómo deben realizarse las comunicaciones entre máquinas, usando XML como lenguaje de intercambio de datos. El mensaje XML en este caso presenta una estructura más compleja de lo habitual debido a que el propio mensaje SOAP va embebido en este mensaje con unas cabeceras y formato únicos y específicos de este protocolo.

A partir del año 2002, aparece un gran rival para SOAP, el que era hasta este momento el protocolo estrella en el campo de los *Web Services*, REST, un estilo arquitectónico que hace uso de diferentes estándares y protocolos para realizar la comunicación con el servicio Web. REST usa el protocolo HTTP para la comunicación entre máquinas a nivel de aplicación, sin utilizar una capa adicional como hace SOAP.

REST se caracteriza por no tener estado. De cara a conseguir servicios escalables, el servidor no conserva el estado de solicitudes anteriores de un determinado cliente. Por ello, el cliente tiene que enviar en cada solicitud todo el estado de su sesión, frecuentemente realizado mediante un *token* que le ayude a “recordar” al servidor.
El hecho de que REST no tenga estado permite una mejor escalabilidad del sistema que junto a la sencillez de acceso al servicio y la variedad del formato de mensaje de intercambio, ha permitido que REST sea considerado como una de las mejores opciones para implantar *Web Services* en estos últimos años. A estos servicios se les denomina APIs REST o servicios Restful.

REST tiende a simplificar las cosas y por ello empieza a ser más frecuente utilizar un lenguaje diferente para representación de los datos, el lenguaje JSON, aunque REST permite el uso también de mensajes XML.

Hoy en día todas las máquinas que están presentes tienen muchas diferencias entre sí. Existen sistemas muy distintos, con sistemas operativos, hardware y diferentes bases de datos integradas. Debido a estas diferencias entre clientes y servidores, la transferencia de datos se realiza mediante lenguajes escritos en archivos de texto plano consiguiendo de esta manera que no importe las características de las máquina para poder comunicarse entre sí.

1.2. Estado del arte de los Web Services

Anteriormente hemos visto someramente las dos tecnologías para crear *Web Services* más en auge del momento, por lo que es importante considerar y decidir qué arquitectura es más apropiada para nuestro sistema y el uso final que le vamos a dar.

Comparar REST y SOAP, quizás no es lo más correcto debido a que no son exactamente lo mismo. Mientras que SOAP es un protocolo estándar definido para realizar esta función, REST es un estilo arquitectónico, por lo que no es un estándar en términos de integración e implementación. REST utiliza varios estándares y protocolos ya definidos para su funcionamiento.

Independientemente de ese detalle, veremos qué nos aporta cada una de ellas, analizando sus ventajas y sus inconvenientes. A la hora de compararlas, analizaremos su uso, flexibilidad, ancho de banda y seguridad:

- **Uso**

A la hora del uso, por norma general REST facilita tanto la creación del servicio como el acceso al mismo y se puede considerar que es más sencillo que SOAP

- **API flexible y simple**

La parte más importante aquí es la simplicidad con la que REST permite el uso del *Web Service*. Todos los recursos creados y funciones disponibles en el servicio se pueden manipular mediante una serie de URIs definidas con anterioridad. De esta manera cualquier cliente o servidor puede acceder al servicio con tan solo una petición HTTP indicando la URI requerida y los parámetros necesarios.
En cambio con SOAP se necesitan unos conocimientos más amplios de XML al tener que realizar la petición con este formato y hacer consultas al WSDL para conocer información del método a invocar.

- **Ancho de banda**

En SOAP, los mensajes que se envían se realizan siguiendo la estructura que define el estándar SOAP y con un código determinado el cual está añadido a un fichero XML. Debido a esto, enviar un mensaje con SOAP es más pesado que enviar cualquier fichero de datos sin estar encapsulado. Con REST, se envía la información necesaria en un fichero XML o JSON, siendo más ligero que el enviado a través de SOAP, el cual se transmite directamente utilizando el protocolo de transporte HTTP.

- **Seguridad**

El tema de la seguridad probablemente es uno de los más importantes a la hora de comparar estas dos opciones.

En el caso de REST la seguridad depende del protocolo de transporte HTTP, heredando sus medidas de seguridad. En cambio SOAP define su propio sistema de seguridad denominado WS-Security.

La seguridad en REST mediante HTTP utilizando SSL aseguraría el mensaje enviado solo mientras se envía, pues utiliza mecanismos de seguridad de la capa de transporte.

Con WS-Security no solo se consigue seguridad del mensaje a nivel de transporte sino a nivel global fijando la seguridad en la capa de aplicación.

Como punto positivo de utilizar SSL sería la mayor rapidez de uso frente a WS-Security.

En definitiva, WS-Security cuenta con medidas para la autenticación, integridad, confidencialidad y no repudio. Utilizando SSL no podríamos asegurar el no repudio.

Otros aspectos a tener en cuenta:

En la tabla 1 se comparan otros puntos interesantes entre REST y SOAP.

REST	SOAP
Orientado a recursos	Orientado a RPC
Servidor no guarda el estado, todo lo hace el cliente	Servidor guarda la sesión
Protocolo HTTP como nivel de aplicación	Utiliza HTTP como túnel para la transmisión de mensajes

Tabla 1. Comparativa básica REST-SOAP

Una vez visto alguno de los puntos más importantes, se podrá visualizar un lista con algunas de las ventajas e inconvenientes de estas dos tecnologías con el fin de ayudarnos a esclarecer cual de ellas puede ser la más ventajosa para nuestro proyecto.

❑ REST

Aspectos positivos

- Fácil de implementar y simple de interpretar
- Es más ligero, sus respuestas contienen exactamente la información que necesitamos. Solo se envía la representación de un recurso. Bajo consumo de recursos
- Al no tener estado, facilita la sincronización de servidores y por consiguiente facilita la escalabilidad del sistema
- Es flexible en cuanto al tipo de respuesta que se necesita, ya que puede consumir XML o JSON
- Funciona muy bien con dispositivos móviles y aplicaciones RIA (*Rich Internet Application*) basadas en Javascript facilitando la portabilidad a otras tecnologías
- HATEOAS (*Hypermedia as the Engine of Application State*), posibilita el descubrir otros recursos cuando recibamos la representación de uno de ellos. En dicha representación, los recursos que vengan definidos, se identifican en forma de hipervínculo para poder localizarlos instantáneamente

Aspectos negativos

- La seguridad puede ser un problema y puede llegar a ser una tarea muy difícil de implementar correctamente
- No hay un estándar en sus respuestas por lo que no se definen tipos de datos
- A la hora del diseño, el manejo del espacio de nombres puede ser algo más rebuscado
- Pocas herramientas de desarrollo
- Menos flexible

SOAP

Aspectos positivos

- Utilizando tecnología .NET o Java es muy sencillo de consumir y existen varias herramientas de desarrollo para poder consumirlo
- Funciona muy bien en entornos empresariales
- Buen manejo de errores
- Estructura más rígida y fuerte
- Mayor seguridad en el transporte de la información

Aspectos negativos

- Una vez implementado, si se desea cambiar algo en el servidor impacta de forma negativa en los clientes ya que estos tienen que hacer muchas modificaciones al código
- Las respuestas son demasiado complejas y difíciles de interpretar si no se tienen las herramientas correctas para hacerlo
- Es complejo y difícil de entender
- Sólo sirve XML
- No aprovecha la infraestructura web como cachés, negociación de formatos
- Depende de la especificación WSDL para poder operar con los procesos

1.3. Objetivos

El presente proyecto tiene como objetivo la realización de un gestor de bibliotecas personales para el almacenamiento de manera organizada de la información relacionada con cualquier libro y facilitar así su consulta y búsqueda.

El servicio Web que se quiere implantar, se requiere que sea lo más ligero posible y fácilmente escalable.

Entre cliente y servidor se podrán intercambiar diferentes tipos de archivos, XML, JSON, etc. En lo que a seguridad respecta, se querrá que la conexión entre cliente y servidor se realice mediante el protocolo HTTPS.

Se ha querido que este servicio tenga una proyección de futuro y se ha desarrollado pensando en que será utilizado desde diferentes plataformas, programadas en diferentes lenguajes de programación.

Por todas estas características requeridas y la motivación de investigar una nueva tecnología, se ha decidido implementar el servicio utilizando el estilo arquitectónico REST.

Para poder realizar un proyecto lo más completo posible y abarcar otros temas de interés se realizarán las siguientes tareas:

- Estudio en profundidad del estilo arquitectónico REST
- Realización del proceso completo de ingeniería del software para el servicio a implementar; SRS, arquitectura del software, diagrama de clases, etc
- Desarrollo e implementación del servicio creando una API REST
- Creación de la base de datos con replicación de datos, almacenadas en diferentes máquinas y siendo independientes tanto del cliente como del servidor para así profundizar en algún concepto de sistemas distribuidos
- Desarrollo completo e implementación de una aplicación web para consumir el servicio web

2. PROTOCOLO HTTP

En este capítulo se hablará de HTTP (*Hypertext Transfer Protocol*), el protocolo más importarte en el funcionamiento de cualquier servicio Web y más si cabe en aquellos que utilizan REST, pues este explota al máximo las características que tiene HTTP.

Como se ha comentado con anterioridad, navegadores, servidores, aplicaciones web y otros servicios utilizan este protocolo para poder transferir cualquier tipo de información y comunicarse entre ellos.

HTTP se encarga de enviar cantidades enormes diariamente de imágenes, páginas Web, películas, audio y otro contenido a través de internet.

Lo mejor de todo es que logra transmitir toda esta gran cantidad de datos de manera:

- Rápida
- Conveniente
- Fiable

Se puede observar que HTTP cuenta con una serie de características muy interesantes pero, no hemos comentado nada sobre la seguridad de nuestros mensajes. ¿Toda la cantidad de tráfico de mensajes que se transmiten diariamente puede ser dañada o alterada? Lo mejor de todo es que no, gracias al uso de lo que denominamos los protocolos de transmisión de datos SEGUROS.

Para entender el protocolo HTTP deberemos profundizar en algunos conceptos preliminares, detallados a continuación.

Clientes Web y servidores

Cualquier contenido Web, se encuentra alojado en servidores Web, los cuales utilizan el protocolo HTTP para comunicarse. Los servidores Web sirven el contenido Web que alojan cuando reciben una petición de un cliente HTTP, como ocurre en la petición mostrada en la figura 6. El cliente Web más conocido es el buscador Web.

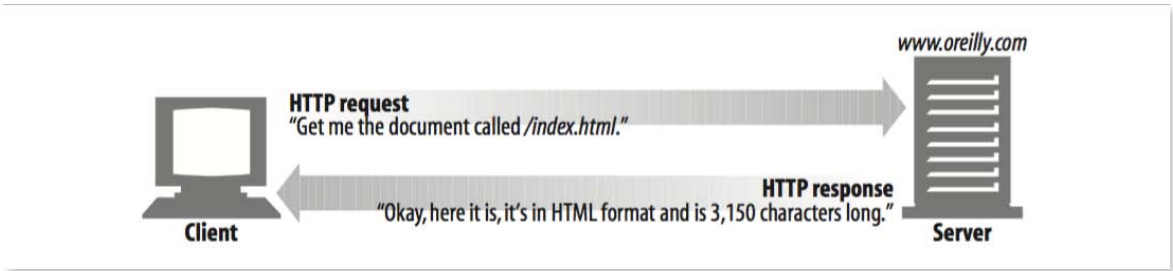


Figura 6. Petición-respuesta HTTP

Recursos

Anteriormente hablábamos del contenido que alojaba un servidor. A todo ese contenido se le denomina recursos. Los recursos pueden ser de dos tipos:

- Ficheros fijos —> Son el tipo más simple. Pueden ser archivos de texto, HTML, PDFs, etc
- Contenido dinámico —> Programas software que generan un contenido

En definitiva podemos definir un recurso, como cualquier contenido que podamos alojar en el servidor. Debido a la cantidad de datos diferentes que pueden ser enviados, se utiliza una etiqueta especial para identificarlos. A esta etiqueta se le denomina MIME (*Multipurpuse Internet Mail Extensions*), la figura 7 muestra un ejemplo de este tipo de información.

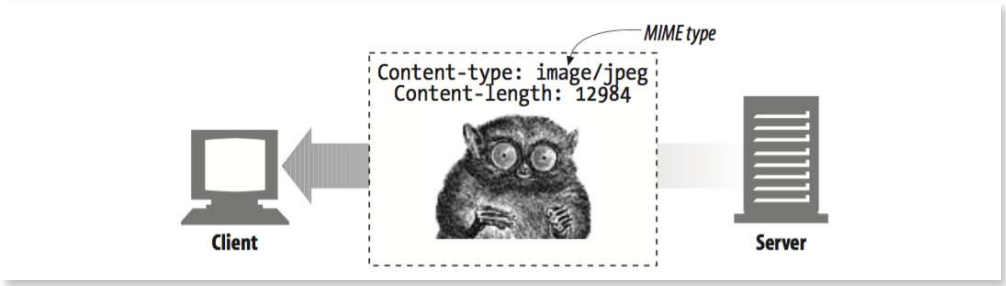


Figura 7. Tipo MIME en mensaje HTTP

En la siguiente lista podemos ver algunos de los valores que pueden tener esta etiqueta:

- text/html
- text/plain
- image/jpeg
- video/quiqktime
- application/vnd.ms-powerpoint

URIs y URLs

El localizador de recursos uniforme o URL (*Uniform Resource Locator*) es la forma más común para localizar un recurso, un ejemplo de este localizador se visualiza en la figura 8. La URL, identifica mediante una cadena de caracteres unos recursos que pueden cambiar, es decir, apunta a recursos variables en el tiempo.

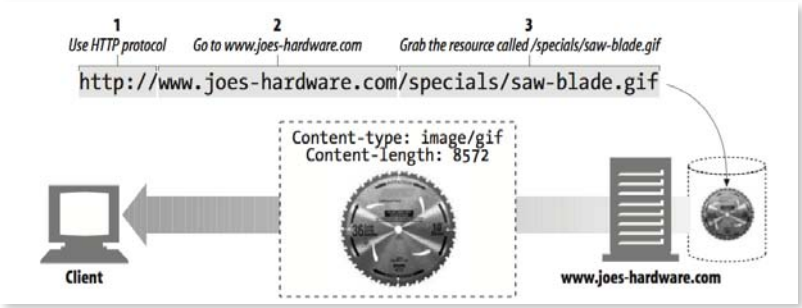


Figura 8. Identificación de un recurso mediante la URL

Cada recurso alojado en el servidor tiene un nombre determinado, por lo que para poder identificar cada uno de los recursos de manera unívoca, se utiliza una cadena de caracteres denominada URI (*Uniform resource identifier*). La URI amplía la información de una URL, pues esta última localiza un recurso pero la URI detalla el nombre de un recurso determinado para poder identificarlo.

Ejemplo: http://www.joes-hardware.com/specials/saw-blade.gif

Para poder realizar una determinada acción existen diferentes métodos y mensaje que se utilizan en la comunicación entre ambos. La comunicación entre el cliente y el servidor se realiza a través de mensajes HTTP y para poder realizar una comunicación entre ellos debe existir primeramente un mensaje de petición por parte del cliente y posteriormente un mensaje de respuesta por parte del servidor.

HTTP soporta diferentes métodos de petición, a los que se les denomina métodos HTTP. Cada petición que realiza el cliente tiene un método asociado. Aunque hablaremos de estos métodos más adelante, los más comunes, se detallan a continuación:

- GET
- PUT
- DELETE
- POST
- HEAD

Cuando el servidor responde al cliente, este mensaje incorpora un código de estado. Este código consta de 3 dígitos que indican al cliente lo sucedido y si se necesita hacer alguna otra operación.
Algunos de los más comunes son los siguientes:

- 200
- 302
- 404

Un ejemplo de la petición del cliente como la respuesta del servidor, se puede visualizar en la figura 9:

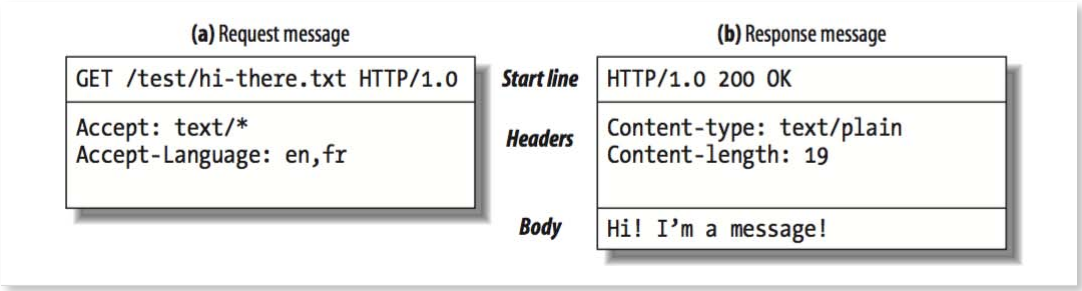


Figura 9. Composición de los mensajes HTTP

Como se puede visualizar en la imagen anterior, los mensajes pueden constar hasta de tres partes diferentes:

- **línea de inicio**: indica el método o el código enviado
- **cabecera**: constan de un par nombre-valor para indicar información sobre la petición o respuesta
- **cuerpo**: en el cuerpo se puede incorporar de manera opcional, cualquier tipo de dato

De manera sencilla se ha visto que contiene mensajes HTTP, pero hasta el momento no se ha hablado de un tema muy importante, ¿cómo se trasladan de un lado a otro? Pues bien, estos mensajes se transmiten a través del protocolo TCP (*Transmisión Control Protocol*).

HTTP es un protocolo de la capa de aplicación, por lo que no se preocupa en absoluto de los detalles de la red de comunicación y deja todo el trabajo a TCP/IP.

Al trabajar bajo TCP, a su vez, tiene las ventajas que nos aporta este protocolo: libre de errores en el transporte de datos. Los datos llegarán en el mismo orden en el que salieron y un flujo de datos no segmentado, permitiendo archivos de cualquier tamaño en cualquier momento.

Entonces, una vez que se ha establecido la conexión TCP, para lo que se necesita la dirección IP y un número de puerto, todos los mensajes que se intercambien entre cliente y servidor nunca se **perderán, dañaran o llegarán en otro orden.**

Por todo esto podemos decir que el protocolo HTTP está encapsulado sobre TCP y de esta manera usa TCP para poder transportar los datos.

A continuación se detalla cómo sería el flujo normal de una conexión HTTP encapsulada sobre TCP y en la figura 10 se detallan los pasos de manera gráfica:

a) El navegador extrae el nombre del servidor de la URL

b) El navegador convierte el nombre del servidor en la dirección IP del servidor

c) El navegador extrae el numero de puerto de la URL, si hay. Por defecto, 80

d) El navegador establece la conexión TCP con el servidor Web

e) El navegador envía un mensaje de petición HTTP al servidor

f) El servidor envía un mensaje de respuesta HTTP al navegador

g) La conexión se cierra y el navegador sirve el documento

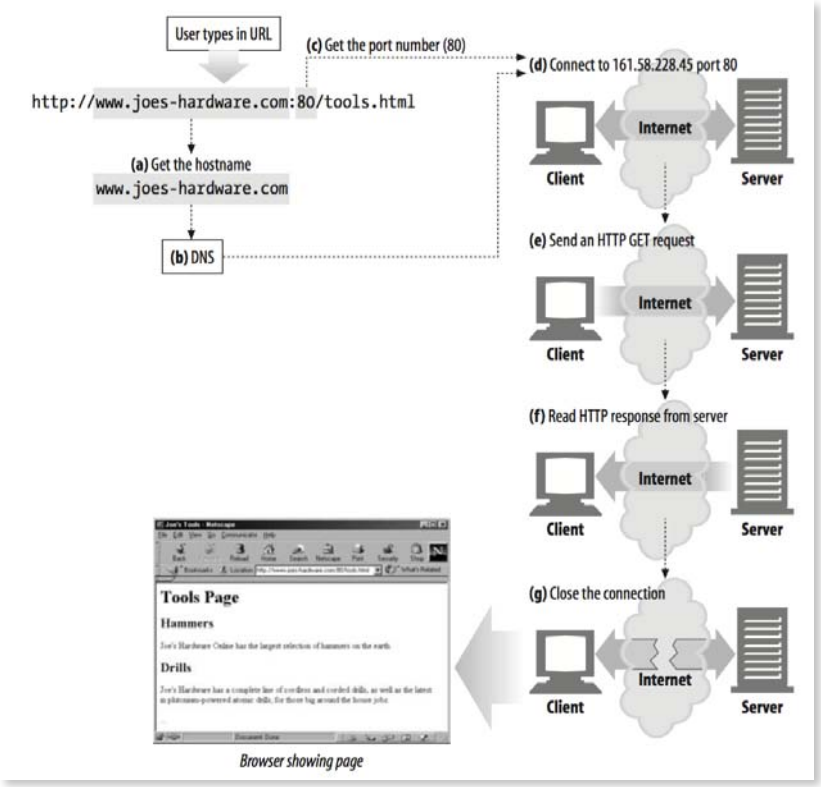


Figura 10. Flujograma conexión HTTP bajo TCP

Por todo lo visto hasta el momento hace pensar que siempre existe una conexión directa entre el cliente y el servidor, pero existen otras aplicación Web que interactúan en esta conexión para poder mejorar la experiencia de usuario, seguridad, etc:

- **Proxys**

  Los servidores proxy, como se puede ver en la figura 11, son muy importantes a nivel de seguridad Web, integración de aplicaciones y optimización del rendimiento. Se encuentran entre cliente y servidor. Reciben y distribuyen todas las peticiones a los servidores.

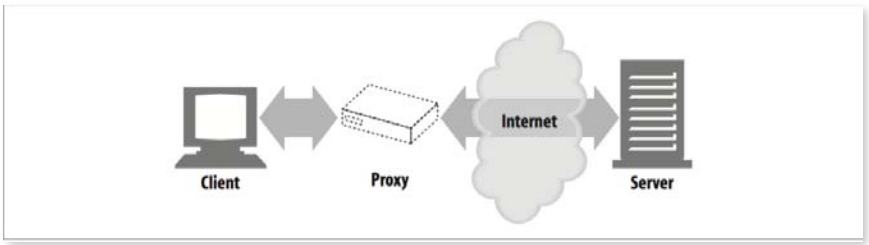


Figura 11. Ejemplo de proxies en la comunicación

- **Cachés**

Son proxies especiales que se encargan de guardar una copia de un recurso popular que pase por ese proxy. De esta manera al recibir otras peticiones, el propio proxy podrá servir ese recurso sin tener que interactuar con el servidor, la figura 12 muestra un ejemplo de estos proxies especiales.

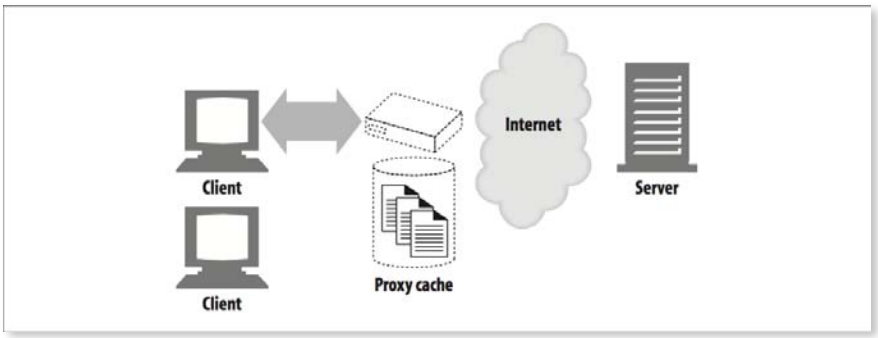


Figura 12. Ejemplo uso de caché

- **Gateways**

Los gateway, reflejados en la figura 13, son servidores especiales que actúan de intermediarios de otros servidores. Normalmente se usa para convertir tráfico HTTP a otro protocolo como por ejemplo, FTP. Reciben las peticiones de clientes como si fuera el servidor original, sin que el cliente se de cuenta de ningún cambio.

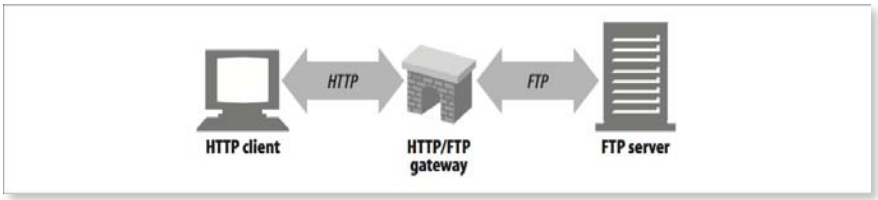


Figura 13. Ejemplo uso de gateways

- **Tunnels**

Los túneles, en la figura 14 se detalla el trafico entre cliente y servidor con un túnel SSL, son aplicaciones HTTP que una vez ejecutadas transportan todo tipos de datos entre dos conexiones. Se usa para poder transportar datos distintos a HTTP bajo conexiones HTTP.
El uso más extendido de los túneles es para transportar tráfico encriptado, SSL (*Secure Rockets Layer*)

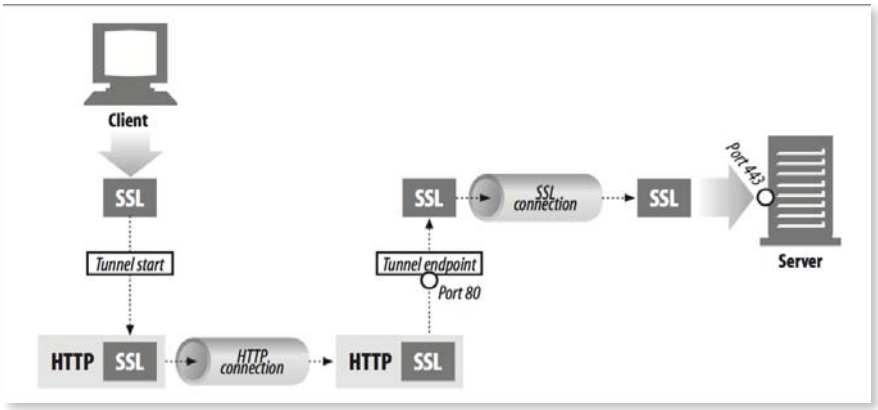


Figura 14. Túneles-SSL

Tras ver unos conceptos esenciales sobre el protocolo HTTP, profundizaremos algo más en alguno de los aspectos vistos.

Ya sabemos que la forma de encontrar, usar y compartir un recurso es utilizando una URL, denominado como el nombre estandarizado de ese recurso en internet.

Para familiarizados con una URL, definamos las partes de la siguiente URL: http://www.joes-hardware.com/seasonal/index-fall.html

- La primera parte de la URL (HTTP) se le denomina esquema, el cual nos indica como vamos a acceder al recurso. En este caso, mediante el protocolo HTTP
- La segunda parte (www.joes-hardware.com) se refiere a la localización del servidor, indicando dónde se encuentra alojado el recurso
- Y la tercera parte (seasonal/index-fall.html) indica la ruta donde se encuentra el recurso

El esquema genérico de una URL es el siguiente:

```
<scheme>://<user>:<password>@<host><port>/<path>;<params>?<query>#<frag>
```

Una vez familiarizados con la estructura de una URL típica, nos damos cuenta que no todos los valores están presentes en las URL que vemos diariamente y eso es porque no todos los valores son obligatorios y dependiendo de lo que queramos realizar se deberán utilizar unos u otros parámetros.

Las partes más importantes son el esquema, el host y la ruta, para poder indicar la forma a la que se quiere acceder a un recurso y cómo acceder a ese recurso.
Aunque los anteriores citados son los más importantes, a continuación se detalla cada uno de los valores que puede haber en una URL.

- **Esquema**:

  Es el principal valor que nos indica cómo vamos a acceder al recurso en cuestión. Indica el protocolo que se necesita para poder realizar una comunicación

  Expresión regular: (a-zA-Z)+:

- **Hosts y puertos:**

  Para acceder a un recurso, primeramente debemos conocer qué maquina aloja el recurso y además, el servicio que tiene acceso a ese recurso dentro de la máquina.

  Con estos dos valores cubrimos esta necesidad. El host, bien como IP, o bien, como *hostname*, nos proporciona la máquina que aloja el servicio y el puerto nos identifica el puerto de red donde el servidor está escuchando para entablar una conexión

- **Nombre de usuario y contraseña:**

  No son unos valores tan conocidos porque no siempre son necesario para poder acceder al recurso, pero algunos servidores requieren estos parámetros para poder acceder. Comúnmente ocurre con servidores FTP, los cuales piden usuario y contraseña antes de acceder

  ftp://anonymous:pwd@ftp.example.edu/pub/gnu

- **Ruta:**

  Este componente de la URL especifica exactamente dónde se encuentra el recurso dentro de la máquina servidor.

  http://www.example.com/seasonal/index-f.html

  En este ejemplo, la ruta que el servidor necesita para poder localizar el recurso pedido en sí, es /seasonal/index-f.html.

- **Parámetros:**

  Por norma general, indicando el *host* y la ruta al recurso es suficiente para obtener un recurso pero hay veces que se necesita detallar algún otro parámetro. Estos nuevos parámetros se pueden indexar al final de la URL.
  Hay veces que no pasar esos parámetros repercute ya no solo en no recibir el recurso pedido sino en recibirlo erróneamente. Esto último ocurre en el caso de utilizar el protocolo FTP, con este protocolo el recurso se puede transferir de dos formas distintas, binario o texto. En caso de no pasar un parámetro indicando cómo queremos que se transmita el recurso puede hacer que recibamos el recurso en un formato no deseado.

  Estos parámetros van al final de la misma URL y constan de dos parámetros nombre/valor y estos están separados del resto de la URL por un punto y coma

  ftp://prep.ai.mit.edu/pub/gnu;type=d

  En este ejemplo podemos visualizar un parámetro, type=d, donde “*type*" equivale al nombre y “d” al valor

- **Querys**

Algunas servicios permiten que se realicen querys a una base de datos directamente desde la URL para comprobar algún dato en concreto. Para esto será necesario pasar los valores de la *query* a realizar a través de parámetros. A diferencia de cómo se pasaban los parámetros normales, en este caso para indicar que lo que se pasa en una *query*, al final de la URL se separa con una marca de interrogación (?). En el ejemplo siguiente se visualiza cómo poder realizar esto:

http://www.example.com/inventory.cgi?item=12345

En este ejemplo se visualiza cómo realizar una búsqueda de un ítem determinado en una base de datos siendo la *query* enviada como parámetros: item=12345

- **Fragmentos**

Algunos recursos, como por ejemplo páginas HTML, pueden estar divididas en diferentes "niveles".

Con los fragmentos podemos referirnos a una parte determinada del recurso indicándolo en la URL. Lo podemos realizar indicándolo con el símbolo almohadilla.

http://www.example.com/index.html#drills

En este ejemplo estaríamos accediendo a una parte especifica denomina *drills* del recurso index.html

De manera introductoria se han visto las partes de las que consta un mensaje HTTP pero ¿qué y cómo son realmente estos mensajes? Son bloques de datos transferidos entre diferentes aplicaciones. Estos inician con un texto de metainformación para describir el contenido y significado del mensaje seguido de diferentes datos opcionales, que veremos más adelante. Estos mensajes son transmitidos entre clientes, servidores y *proxies*, la transmisión de estos imagen se puede ver en la imagen 15.

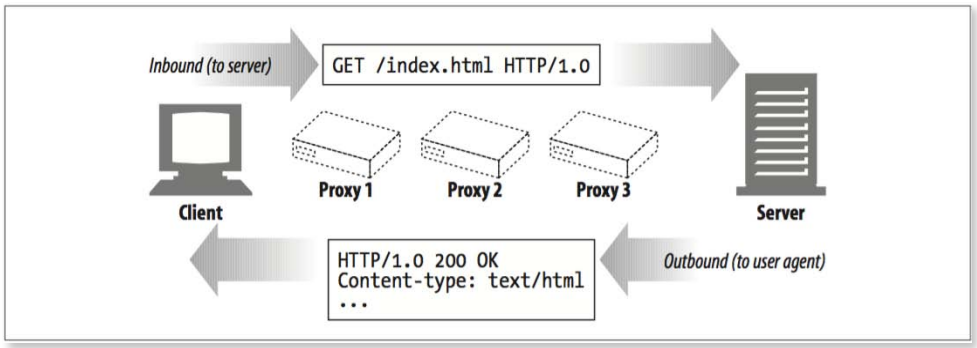


Figura 15. Comunicación cliente-servidor con intermediarios

Estos mensajes son simples y están formados por tres partes, detalladas a continuación y de manera más precisa en la figura 16:

-Línea inicial: se encarga de describir el mensaje
-Conjunto de cabeceras: contiene diferentes atributos
-Cuerpo: campo opcional que contiene datos

Tanto la línea inicial como las cabeceras son texto ASCII separados entre líneas. En cambio el cuerpo puede contener texto, datos en binario o incluso puede ir vacío.

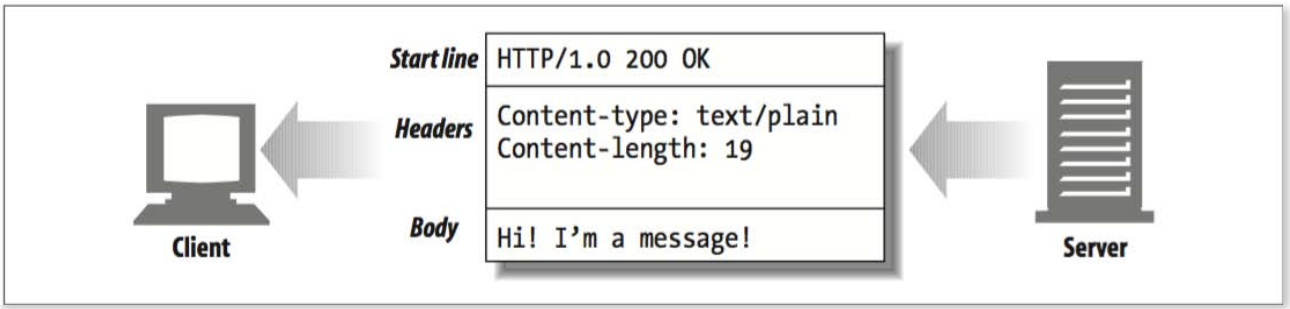


Figura 16. Composición mensaje HTTP

Los dos tipos de mensajes existentes son los de petición y los de respuesta, en los ejemplos se muestran la estructura de cada uno de ellos

Esquema mensaje de petición:

<method> <request-URL> <version>
<headers>

<entity-body>

Esquema mensaje de respuesta:

<version> <status> <reason-phrase>
<headers>

<entity-body>

Las partes que conforman un mensaje HTTP, referidas anteriormente, se ven con más detalle a continuación:

- **Línea de inicio**

Todos los mensajes HTTP comienzan con esta línea, la cual dependiendo del mensaje, tendrá unos parámetros u otros.

El parámetro común de los dos tipos de mensaje es la **versión,** indicando la versión del protocolo HTTP usado. Tiene el siguiente formato, HTTP/x.y. e indica de esta manera lo que cada parte de la comunicación, clientes y servidores, pueden hacer y qué formato de mensaje pueden entender.
La versión indica siempre el protocolo más alto que se soporta.
La Version más reciente es HTTP/2.

Los siguientes parámetros de los que vamos a hablar dependen del tipo de mensaje.

✳ **Mensajes de petición**

-**Método**:

Es el atributo con el que empieza esta línea para decir al servidor qué hacer.
Algunos de los métodos más usados son los siguientes:

GET

Es el comando más común. Suele ser usado para solicitar a un servidor obtener un recurso. En caso de uso en aplicación se desaconseja su uso por seguridad, pues es un método que envía parámetros a través de la URL y son fáciles de captar

HEAD

Es un método que se comporta como el GET pero el servidor no envía ningún cuerpo. Solo envía las cabeceras del mensaje. Se debe comprobar que las cabeceras que se devuelven son las mismas que se devolverían usando el método GET.
El mayor uso que se le da es para obtener información de un recurso sin conseguirlo, ver si un recurso existe o ha sido modificado

PUT

Este método permite subir, cargar, actualizar documentos de un servidor.
Algunos servidores debido a que con este método se pueden modificar cosas del servidor ponen un paso de autenticación previo

POST

Método utilizado para enviar entradas de datos al servidor y que sean procesados por el recurso identificado. Hoy en día es frecuentemente usado en formularios HTML

TRACE

Este método solicita al servidor que en la respuesta meta todos los datos que reciba en el mensaje de petición. Se utiliza con fines de depuración y diagnóstico ya que el cliente puede ver lo que llega al servidor

OPTIONS

Este método pide al servidor sobre información acerca de las diferentes prestaciones del servicio Web, de esta manera podemos obtener los métodos soportados

DELETE

Método con el cual se puede borrar el recurso indicado

-**URL de petición**:

Es una URL completa indicando el recurso del servidor en el que se quiere ejecutar el método indicado

✳ **Mensaje de respuesta**

-Código de estado:

Estos códigos constan de tres dígitos y sirven para indicar el resultado de la operación por parte del servidor.
Estos se clasifican en cinco grandes campos:

- ✦ 100-199: Códigos de estado de información, detallados en la tabla 2
- ✦ 200-299: Códigos de estado de éxito, detallados en la tabla 2

Código	Mensaje	Descripción
10x	**Mensaje de información**	**Estos códigos no se utilizan en la versión 1.0 del protocolo**
20x	**Éxito**	**Estos códigos indican la correcta ejecución de la transacción**
200	OK	La solicitud se llevó a cabo de manera correcta
201	CREATED	Sigue a un comando POST e indica el éxito, la parte restante del cuerpo indica la dirección URL donde se ubicará el documento creado recientemente.
202	ACCEPTED	La solicitud ha sido aceptada, pero el procedimiento que sigue no se ha llevado a cabo
203	PARTIAL INFORMATION	Cuando se recibe este código en respuesta a un comando de GET indica que la respuesta no está completa.
204	NO RESPONSE	El servidor ha recibido la solicitud, pero no hay información de respuesta
205	RESET CONTENT	El servidor le indica al navegador que borre el contenido en los campos de un formulario
206	PARTIAL CONTENT	Es una respuesta a una solicitud que consiste en el encabezado *range*. El servidor debe indicar el encabezado *content-Range*

Tabla 2. Descripción mensajes HTTP 10x y 20x

- 300-399: Códigos de estado de redirección, los más importantes se reflejan en la tabla 3

30x	Redirección	Estos códigos indican que el recurso ya no se encuentra en la ubicación especificada
301	MOVED	Los datos solicitados han sido transferidos a una nueva dirección
302	FOUND	Los datos solicitados se encuentran en una nueva dirección URL, pero, no obstante, pueden haber sido trasladados
303	METHOD	Significa que el cliente debe intentarlo con una nueva dirección; es preferible que intente con otro método en vez de GET
304	NOT MODIFIED	Si el cliente llevó a cabo un comando GET condicional (con la solicitud relativa a si el documento ha sido modificado desde la última vez) y el documento no ha sido modificado, este código se envía como respuesta.

Tabla 3. Descripción mensajes HTTP 30x

- 400-499: Códigos de estado por error del cliente, reflejados en la tabla 4
- 500-599: Códigos de estado por error del servidor, los más importantes se ven en la tabla 5

40x	Error debido al cliente	Estos códigos indican que la solicitud es incorrecta
400	BAD REQUEST	La sintaxis de la solicitud se encuentra formulada de manera errónea o es imposible de responder
401	UNAUTHORIZED	Los parámetros del mensaje aportan las especificaciones de formularios de autorización que se admiten. El cliente debe reformular la solicitud con los datos de autorización correctos
402	PAYMENT REQUIRED	El cliente debe reformular la solicitud con los datos de pago correctos
403	FORBIDDEN	El acceso al recurso simplemente se deniega
404	NOT FOUND	Un clásico. El servidor no halló nada en la dirección especificada. Se ha abandonado sin dejar una dirección para redireccionar... :)

Tabla 4. Descripción mensajes HTTP 40x

50x	Error debido al servidor	Estos códigos indican que existe un error interno en el servidor
500	INTERNAL ERROR	El servidor encontró una condición inesperada que le impide seguir con la solicitud (una de esas cosas que les suceden a los servidores...)
501	NOT IMPLEMENTED	El servidor no admite el servicio solicitado (no puede saberlo todo...)
502	BAD GATEWAY	El servidor que actúa como una puerta de enlace o proxy ha recibido una respuesta no válida del servidor al que intenta acceder
503	SERVICE UNAVAILABLE	El servidor no puede responder en ese momento debido a que se encuentra congestionado (todas las líneas de comunicación se encuentran congestionadas, inténtelo de nuevo más adelante)
504	GATEWAY TIMEOUT	La respuesta del servidor ha llevado demasiado tiempo en relación al tiempo de espera que la puerta de enlace podía admitir (excedió el tiempo asignado...)

Tabla 5. Descripción mensajes HTTP 50x

-Frase-razón:

Frase que indica de manera legible para las personas humanas lo que significa cualquier código de estado.

Ejemplo: “HTTP/1.0 200 NOT OK” —> not ok es el significado del código de estado 200

- **Cabeceras**

Son metadatos que se envían en todos los mensajes tanto de petición como de respuesta que proporcionan información esencial de la transición.

Estas cabeceras aportan una gran flexibilidad al protocolo permitiendo añadir nuevas funcionalidades.

Todas las cabeceras, algunas reflejadas en la tabla 6 y 7, las podemos agrupar en cinco grandes grupos:

- ✦ Cabeceras generales
- ✦ Cabeceras de petición
- ✦ Cabeceras de respuesta
- ✦ Cabeceras de entidad
- ✦ Cabeceras de extensión

Nombre del encabezado	Descripción
Accept	Tipo de contenido aceptado por el navegador (por ejemplo, *texto/html*). Consulte Tipos de MIME
Accept-Charset	Juego de caracteres que el navegador espera
Accept-Encoding	Codificación de datos que el navegador acepta
Accept-Language	Idioma que el navegador espera (de forma predeterminada, inglés)
Authorization	Identificación del navegador en el servidor
Content-Encoding	Tipo de codificación para el cuerpo de la solicitud
Content-Language	Tipo de idioma en el cuerpo de la solicitud
Content-Length	Extensión del cuerpo de la solicitud
Content-Type	Tipo de contenido del cuerpo de la solicitud (por ejemplo, *texto/html*). Consulte Tipos de MIME
Date	Fecha en que comienza la transferencia de datos
Forwarded	Utilizado por equipos intermediarios entre el navegador y el servidor
From	Permite especificar la dirección de correo electrónico del cliente
From	Permite especificar que debe enviarse el documento si ha sido modificado desde una fecha en particular
Link	Vínculo entre dos direcciones URL
Orig-URL	Dirección URL donde se originó la solicitud
Referer	Dirección URL desde la cual se realizó la solicitud
User-Agent	Cadena con información sobre el cliente, por ejemplo, el nombre y la versión del navegador y el sistema operativo

Tabla 6. Descripción cabeceras HTTP 1

Nombre del encabezado	Descripción
Content-Encoding	Tipo de codificación para el cuerpo de la respuesta
Content-Language	Tipo de idioma en el cuerpo de la respuesta
Content-Length	Extensión del cuerpo de la respuesta
Content-Type	Tipo de contenido del cuerpo de la respuesta (por ejemplo, *texto/html*). Consulte Tipos de MIME
Date	Fecha en que comienza la transferencia de datos
Expires	Fecha límite de uso de los datos
Forwarded	Utilizado por equipos intermediarios entre el navegador y el servidor
Location	Redireccionamiento a una nueva dirección URL asociada con el documento
Server	Características del servidor que envió la respuesta

Tabla 7. Descripción cabeceras HTTP 2

- **Cuerpo**. Contiene el documento solicitado en caso de ser la respuesta. En caso de petición, el cuerpo es un conjunto de líneas opcionales y espacio donde se envían parámetros de un formulario enviado por POST

3. ESTILO ARQUITECTÓNICO REST

A lo largo de los capítulos anteriores se han visto algunos conceptos y características de REST, pero en este capítulo ahondaremos en los detallen que lo definen. A su vez, se podrá visualizar de manera práctica la creación y funcionamiento de este nuevo estilo.

REST fue definido por Roy Fielding en su tesis doctoral en el año 2002. En ese momento, Roy se cuestionaba tres aspectos:

- ¿Por qué la Web es tan predominante?
- ¿Por qué la Web es tan escalable?
- ¿Cómo podríamos aplicar la arquitectura de la Web en nuestros proyectos?

Estas preguntas son tan importantes porque explican el funcionamiento de la Web. Tras conocer su funcionamiento y todos los aspectos positivos que aportaba propició que Roy empezara a plantearse el poner en práctica estas mismas características en el desarrollo de servicios Web.

Al mismo tiempo existen un par de preguntas que todos los los servicios Web actuales responden de diferente manera. La respuesta a ellas ayudará a saber cómo funcionan estos servicios en la Web.

- ¿Cómo sabe el servidor qué hacer con los datos recibidos?
- ¿Cómo sabe el servidor el recurso exacto con el que tiene que operar?

Todas estas cuestiones planteadas aportan un conjunto de diferentes criterios de diseño. Todos estos criterios caracterizan a un estilo arquitectónico específico denominado ROA (*Resource-Oriented Arquitecture*), o lo que es lo mismo, una arquitectura orientada a los recursos y no a procedimientos como otros servicios Web.
Definimos un recurso como cualquier cosa que pueda ser almacenada en un ordenador y representarlo mediante un flujo de bits como puede ser un documento, el resultado de un algoritmo, información de una base de datos, etc.

Con el tiempo a este estilo arquitectónico se le ha acabado llamando REST, cuyas principales propiedades son las siguientes:

- **<u>Capacidad de direccionamiento</u>**

  Se considera que una aplicación es direccionable, capaz de ser llamada a través de una dirección, si expone todo el conjunto de datos que posee como recursos y cada uno de estos está accesible a través de un único identificador o ruta, una URI.
  Esto significa que todos los recursos están accesibles mediante una determinada dirección, pudiendo así existir un numero infinito de URIs para un servicio.
  De esta forma enviando alguna URI definida en nuestro servicio, mediante una solicitud HTTP, tendremos acceso a todos los recursos disponibles.

- **<u>Interfaz limitada y uniforme</u>**

En la Web realmente solo existen unas pocas acciones que podemos realizar con un recurso. HTTP provee cuatro métodos para realizar las operaciones más comunes:

- Obtener una representación de un recurso: HTTP GET
- Crear un nuevo recurso: HTTP PUT o HTTP POST
- Modificar un recurso: HTTP PUT
- Eliminar un recurso: HTTP DELETE

La idea de este principio es utilizar un conjunto finito de operaciones del protocolo de aplicación del servicio distribuido. Esto quiere decir que no tenemos un parámetro de “acción” especificado en la URI cada actividad a realizar sino que hacemos uso solo de los métodos del protocolo HTTP.
Los métodos más comunes son GET, PUT, DELETE, POST, HEAD y OPTIONS y son los responsables de indicar la función a realizar sobre el servicio.
Existen otros métodos para conseguir realizar cualquier tipo de operación con REST, pero la mayoría de las actividades se realizan con estos pocos métodos.

A continuación se detalla el funcionamiento de los métodos más importantes:

- **GET**
  Para obtener un recurso se debe realizar una petición usando el comando GET sobre la URI de ese recurso. Esta acción no cambia el estado del mismo.
  En el caso de una petición GET, el servidor devuelve una representación del recurso requerido en el campo entity-body.

- **POST**
  Junto con el método GET, otro de los métodos más usados es el método POST, encargado de crear un nuevo recurso pero también de modificarlo.
  El método POST no es idempotente por lo que siempre que se envíe una petición se crea o modifica un recurso en el servidor, aunque esa misma petición ya haya sido enviada con anterioridad.
  Es un método que se debe evitar usar con REST pero que en el caso de envío de datos a través de un formulario será inevitable su uso.

- **PUT**
  Para crear o modificar un recurso, se envía una petición PUT que normalmente incluye un cuerpo en el mensaje. El cuerpo incluye la nueva representación del recurso.
  La principal diferencia respecto al método POST es que es idempotente, en este caso siempre que se realice esta petición con los mismos valores, debe generar el mismo resultado. Por esto, es el mejor método para actualizar un recurso.

- **DELETE**
  Si enviamos una petición DELETE, se elimina un recurso. En este caso el servidor devuelve un mensaje de estado o puede no devolver nada.

Otros métodos que aunque no sean tan utilizados los podemos considerar parte de la interfaz uniforme son:

- **HEAD**
  Con este método se puede obtener los metadatos de un recurso sin tener que descargar todo el cuerpo.
  El uso que se le puede dar es para comprobar si un recurso existe pero sin obtener su representación.

- **OPTIONS**
  Permite al cliente descubrir qué está permitido hacer a un recurso. La respuesta a esta petición contiene la cabecera HTTP Allowheader, indicando así todas las opciones disponibles sobre un recurso.

¿Por qué es importante una interfaz uniforme?

Limitar la interfaz tiene más ventajas que inconvenientes:

-Familiaridad

Lo más importante de REST no es que tenga una interfaz uniforme definida con métodos HTTP porque estos, no serán la mejor opción siempre. Lo más interesante es que REST especifica una interfaz uniforme pero sin detallar cuál. Esto es importante porque adaptándose siempre a lo necesario en cada momento, se consigue uniformidad: en estos momentos todos los servicios usan la interfaz HTTP de la misma forma.
Si no existiera esta interfaz uniforme obtendríamos muchísimos métodos distintos haciendo lo mismo que realiza GET, como por ejemplo doSearch, getPage, nextPrime, etc.
Con la interfaz uniforme conseguimos que todas las opciones posibles sean finitas y bien conocidas por todos los servicios.

-Interoperabilidad

Si el servicio Web está expuesto sobre HTTP existe una alta probabilidad de que los usuarios que quieran usar el servicio puedan hacerlo sin ningún requisito adicional más allá de ser capaces de intercambiar un determinado formato de datos que el servicio ofrece, sin importar el lenguaje de programación de cada uno o cualquier otra característica.
Con REST solo se debe entender el formato de los datos que el servidor devuelve, porque lo que se intercambia son representaciones de un recurso.

-Escalabilidad

Como REST usa un número restringido de métodos bien definidos, esto conlleva a tener un comportamiento predecible a la vez de un increíble rendimiento.
El método GET es el mejor ejemplo. Cuando pedimos una página por segunda vez nos damos cuenta que el tiempo de carga es menor. Esto es debido al uso de un método de lectura que a su vez es seguro e idempotente por lo que los *proxys* Web o incluso el propio navegador pueden guardar la respuesta del servidor evitando así mucho tráfico de información.

- **Orientado a la representación**

El servicio debe estar "orientado a la representación". Lo que se quiere explicar con este concepto es que cada recurso está accesible a través de una URI y de esta manera tener acceso al mismo pero, los recursos no son los datos sino una idea materializada en el momento de diseño de cómo convertir esos datos en un determinado formato. Este formato es a lo que se le llama la representación de un recurso y es transmitido en cada comunicación existente entre el cliente y el servidor.
Un recurso es una fuente de representaciones donde una representación son los datos de un recurso en su estado actual.

En un sistema Restful, la complejidad de la interacción entre el cliente y el servidor, viene definida según las representaciones que se intercambien. Estas representaciones pueden ser cualquier tipo de información útil acerca del estado de un recurso como pueden ser una página web, una imagen, texto plano, XML, JSON, etc.

Con HTTP, la representación es el cuerpo del mensaje de la solicitud o de la respuesta. El cuerpo del mensaje puede estar en cualquier formato que tanto el cliente como el servidor quieran intercambiar. Para poder interpretarlo, se usa el campo '*content-type*' indicando de esta manera el formato a utilizar.

- **Comunicación sin estado**

Esta característica no quiere decir que la aplicación no pueda tener estado, pero si se realiza un servicio Web siguiendo el estilo REST, toda la información que necesita el servidor debe ser enviada en cada petición HTTP.

El servidor solo graba y gestiona el estado de los recursos que expone. Si se necesita un dato de sesión específico, este deberá alojarse y mantenerse en la parte del cliente y se enviará al servidor mediante la "*request*" pertinente. Por decirlo de otra forma, posibles estados de un servidor se deben considerar como recursos y tener así su propia URI (http://miServicio/pagina/2, recordando al servidor la página actual) la cual se envía en la petición HTTP para que el servidor lo procese recibiendo así el estado necesario.

Para algunas cosas parece mejor tener aplicación con estado pues no es necesario enviar peticiones constantemente recordando al servidor donde se encontraba anteriormente pero, solo sería positivo si habláramos de peticiones HTTP individuales. Pero en términos del protocolo en sí sería mucho más complicado.
La eliminación del estado en un protocolo además elimina muchos posibles fallos como pueden ser: el time out del cliente, facilita que una aplicación pueda estar distribuida por balanceadores de carga y también es más fácil el manejo de caché.

Esta es una propiedad muy importante ya que un servicio que no tiene que mantener las sesiones de los clientes, es un servicio más sencillo de escalar, debido a que lo único que hay que hacer es añadir máquinas sin tener en cuenta ninguna información sobre ellas.

- **HATEOAS (*Hypermedia As The Engine Of Application State*)**

A menudo a Internet se le llama la Web porque toda la información se transmite indexando hipervínculos, los cuales, crean otros enlaces con otras paginas Web en Internet. Esto es el concepto principal de este principio arquitectónico.
HATEOAS describe cómo crear enlaces a todos los recursos de un servicio y enviarlos de manera automática en las representaciones transmitidas en cada comunicación sin que el cliente deba realizar ninguna petición previa para consultar otros datos.

Las ventajas de usar HATEOAS en los *Web Services* son las siguientes:

- Transparencia de la localización
- Desacoplamiento de datos de interacción
- Reducción de los errores de transición de estado

```
{
    "id": 78,
    "nombre": "Juan",
    "apellido": "García",
    "coches": [
        {
            "id": 1033
        },
        {
            "id": 3889
        }
    ]
}
```

```
{
    "id": 78,
    "nombre": "Juan",
    "apellido": "García",
    "coches": [
        {
            "coche": "http://miservidor/concesionario/api/
v1/clientes/78/coches/1033"
        },
        {
            "coche": "http://miservidor/concesionario/api/
v1/clientes/78/coches/3889"
        }
    ]
}
```

Como se observa en estos ejemplos, en los dos se recibe la misma representación de un recurso pero a diferencia del primer ejemplo, el segundo a su vez añade unas URIs indicando así cómo acceder a los nuevos recursos que se envían en la representación. El segundo ejemplo muestra el uso de HATEOAS.

Ejemplo de una llamada básica entre un cliente y un servicio REST

Como se sabe, la comunicación entre el cliente y el servidor es mediante peticiones y respuestas HTTP, reflejamos estos mensajes de nuevo en la figura 17.

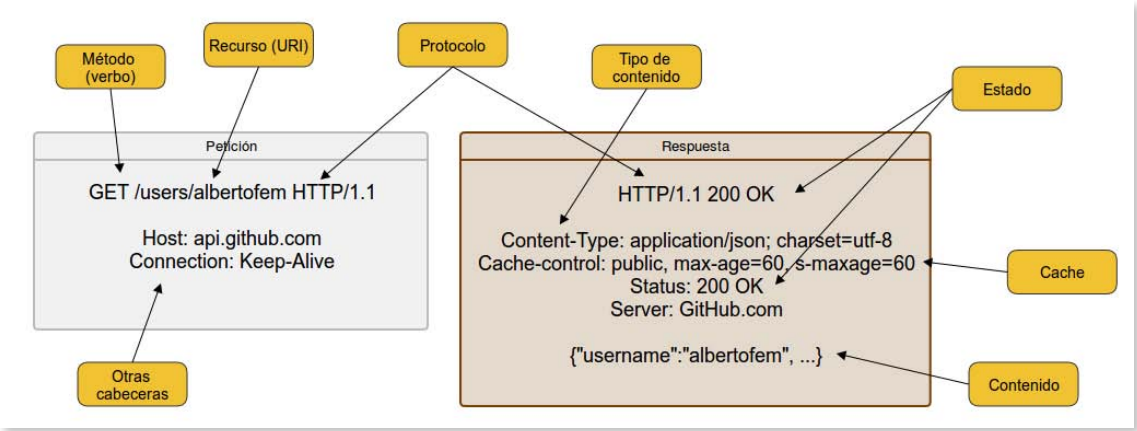


Figura 17. Petición-respuesta HTTP

En la petición se especifica el método determinado (GET, PUT, etc) indicando qué acción queremos realizar sobre un recurso del servicio, reverenciado mediante la URI.
En este ejemplo se realiza un GET sobre un usurario determinado, obteniendo de esta manera la información del usuario.

La respuesta, se encarga de transmitir un mensaje con el resultado de la operación utilizando los mensajes de resultados propios del protocolo HTTP (En este ejemplo: 200 OK). A esta respuesta se le adjunta una representación del recurso pedido en caso de haber sido requerida. En el ejemplo la representación del recurso se realiza intercambiando un archivo JSON con esa información.

Por lo tanto hay que tener claro que el concepto básico de REST es que se interactúa con recursos y no con objetos específicos y estas interacciones son especificadas a través de URIs que apuntan a los recursos definidos. Pongamos un ejemplo:

Se define el recurso genérico video, porque este a su vez se puede desglosar en otros recursos apuntando con mayor precision. A continuación definiremos una serie de URIs que apuntarán a diferentes tipos de recursos relacionados con videos:

- /video —> Raíz de este recurso, más adelante se detalla la función del mismo
- /video/{id} —> Se envía en la URI un identificador de un video
- /video/{id}/{duración} —> Se transmite un identificador y la duración deseada

Podemos definir cualquier URI todo lo precisa que queramos para definir un recurso.
Cuantos más recursos identificados y detallados, más versátil será la API REST desarrollada.

Posteriormente, con los diferentes métodos HTTP se indica la función a realizar sobre esos recursos:

- **GET /video** —> Pedimos obtener una representación mostrando todos los vídeos existentes

- **GET /video/1** —> Pedimos obtener una representación del vídeo número uno

- **POST /video/1** —> En este ejemplo se utiliza la misma URI. Se cambia únicamente el método. De esta manera utilizando un mismo identificador podemos realizar una función diferente. En este caso, se realiza una modificación del video número uno

- **DELETE /video/1** —> del mismo modo que con el método POST, en este caso borramos el recurso vídeo uno

-Ejemplo genérico de interacción entre clientes y servicio REST

En la figura 18 se muestra cómo es la interacción con un servicio REST

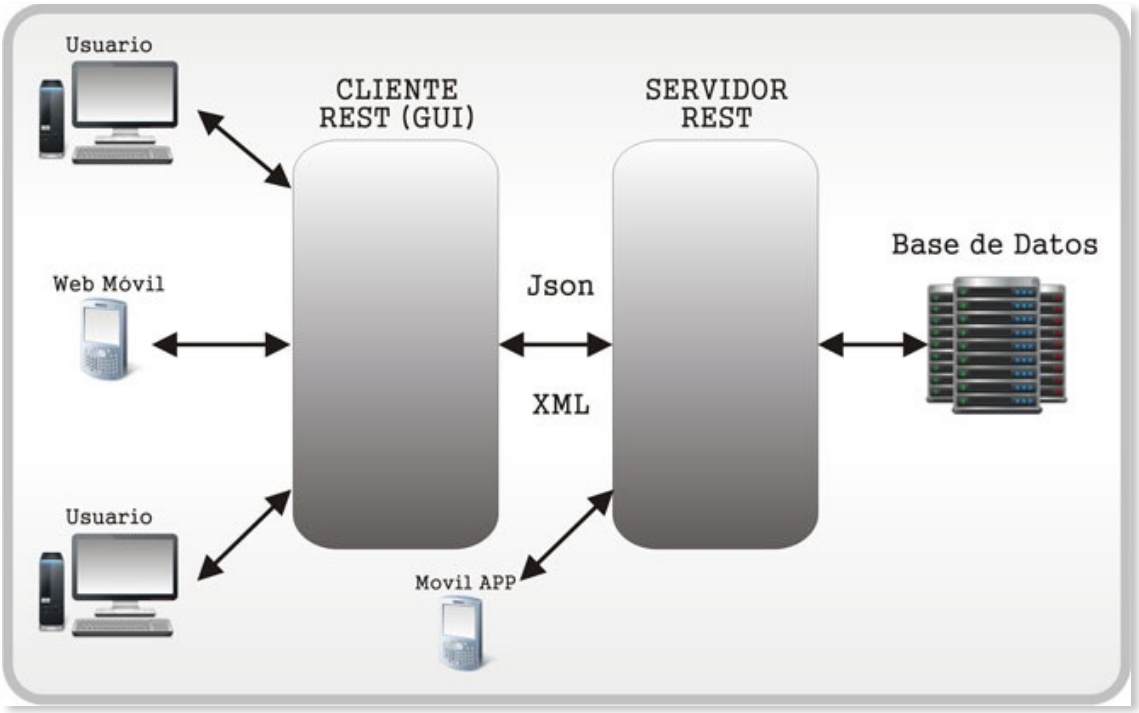


Figura 18. Interacción con servicio REST

Una confusión bastante frecuente y más tras ver estos ejemplos es pensar que REST solo sirve para hacer CRUD (*Create, Read, Update, Delete*) a una base de datos o hacernos creer que una base de datos es un sistema REST pero, no lo es. En algunos aspectos puede ser parecido pero existen algunas diferencias importantes entre llamadas directas a una base datos y BBDD bajo la arquitectura REST.
Si se trata directamente con la base de datos es el propio cliente quien debe conectarse a esa base de datos y realizar las operaciones necesarias.
Si atacamos a un servicio web REST, el cliente mediante un identificador (URI) indica al servicio la función a realizar y es el servicio el que se encarga de hacer las consultas necesarias, favoreciendo de esta manera la conexión de cualquier tipo de aplicación con el servicio y la base de datos de manera más sencilla y despreocupando al cliente de realizar estas operaciones.
Como sabemos, REST permite un intercambio de diferentes tipos de representaciones e información de recursos, xml, pdf, png, gif, excel, etc. Hay pocas BBDD que puedan transmitir esta variedad de

información. Además, un servicio REST puede realizar una serie de operaciones muy diversas como cálculos matemáticos, algoritmos, búsqueda de información, etc y enviar el resultado en un formato determinado permitiendo de esta manera realizar muchas más operaciones muy diferentes a consultas de base de datos.

Para finalizar este capítulo y con el objetivo de matizar algunos detalles de REST, en el anexo a se muestra una conversación que Ryan Tomayko tuvo con su esposa a la que intentó explicar qué es REST. Con el tiempo se ha hecho muy conocida por ser una explicación que trata de manera muy amena y descriptiva el tema que nos atañe.

4. SERVICIO DE BIBLIOTECA BASADO EN WEB SERVICES

4.1.SRS

4.1.1. Introducción

4.1.1.1. Propósito

Este documento especifica los requisitos software de una aplicación que permite la gestión de bibliotecas y todo su contenido, manteniéndolo ordenado y accesible de manera rápida y sencilla.

El contenido de este documento está destinado a ingenieros de casos de uso, analistas, personal de pruebas y mantenimiento y a todos aquellos usuarios de la aplicación.

4.1.1.2. Ámbito

Este documento contiene la especificación de requisitos software (SRS) de la aplicación "Mi Biblioteca". Esta aplicación permitirá crear a los usuarios sus propias bibliotecas virtuales, teniendo de esta manera todo su contenido perfectamente organizado. La aplicación permitirá introducir, eliminar, modificar y visualizar información de los libros añadidos, además se podrá detallar los libros leídos y añadirle una puntuación, si se desea.

La aplicación en ningún momento podrá reproducir el contenido del libro.

4.1.1.3. Definiciones, acrónimos y abreviaturas

API	Application Interface
HTML	HyperText Markup Language
HTMLS	HyperText Markup Language Secure
JSON	JavaScript Object Notation
PHP	PHP Hypertext Pre-processor
MySQL	Structured Query Language
OAuth	Open Authorization
REST	Representational State Transfer
SRS	Software Requirements Specification

4.1.1.4. Referencias

Guía del Usuario de “mi biblioteca” [en construcción].
IEEE (1998). IEEE Recommended practice for software requirements specifications.

4.1.1.5. Visión general del documento

El resto de este documento está organizado como sigue. La sección 2 contiene una descripción global del producto en cuanto a funcionalidad general, restricciones, características del usuario, etc. En la sección 3 se describen con precisión los requisitos de la aplicación "Mi Biblioteca" organizados por “características” tal y como describe el formato A.5 del anexo A del estándar 830-1998 de IEEE (IEEE, 1998).

4.1.2. Descripción general

4.1.2.1.Perspectiva del producto

Esta aplicación es independiente y totalmente autónoma en lo que a la gestión de la biblioteca se refiere y la autenticación para el acceso.

4.1.2.1.1. Interfaces del sistema

La aplicación hará uso del Servicio Web implementado para gestión de bibliotecas. El intercambio de mensajes entre la aplicación y dicho servicio se realizará mediante archivos JSON y el envío de algún tipo de información hacia el servicio se realizará a través de formularios.

4.1.2.1.2. Interfaces de usuario

El usuario interactúa con la aplicación a través del teclado y el ratón. La aplicación tendrá un estilo *responsive*, permitiendo la correcta visualización independientemente del terminal en el que se visualice.

La interfaz presentada consta de menús en la parte superior para dirigirse a las diferentes opciones de la aplicación, botones, campos de entrada y de salida. Los mensajes de error se visualizarán mediante una ventana emergente en la que existirá un botón con el texto “aceptar”, o bien, mediante texto en los campos de salida.

4.1.2.1.3. Interfaces hardware

N/A

4.1.2.1.4. Interfaces software

La aplicación se ejecutará en cualquier sistema operativo y en cualquier navegador disponible. Precisará tener Java 8 y flash 20.0.0.286 instalado en el sistema.

4.1.2.1.5. Interfaces de comunicación

Para la conexión entre cliente y servidor se utilizará un servicio web REST, ya que al soportar múltiples plataformas, permiten una mejor escalabilidad y mantenibilidad.

Para poder hacer uso de esta tecnología, la comunicación se realizará a través del puerto 80 utilizando el protocolo HTTP.

4.1.2.1.6. Restricciones de memoria

El equipo servidor, ya que maneja múltiples conexiones de los clientes, deberá contar con al menos 8GB de RAM.

4.1.2.1.7. Operaciones

N/A

4.1.2.1.8. Requisitos de despliegue

N/A

4.1.2.2. Funciones del producto

Las principales funciones que soportará la aplicación serán:

- Autenticación de manera segura mediante token JWT firmado
- Creación de diferentes bibliotecas personales
- Añadir libros a una biblioteca determinada
- Editar la información de libros existentes en una biblioteca
- Eliminar un libro de una biblioteca
- Gestión del perfil

4.1.2.3. Características del usuario

El usuario no requiere ningún conocimiento adicional para poder manejar la aplicación.

4.1.2.4. Restricciones

El servicio Web estará desarrollado en Java, almacenando los datos en una base de datos MySQL.
La parte del cliente Web estará programado en HTML, JavaScripy y PHP utilizando cURL para la conexión con el servicio Web.
Se utilizará JSON para el intercambio de información entre el cliente y el servidor.
La autenticación tanto en la API como en el sistema se realizará a través de tokens de sesión.

4.1.2.5. Suposiciones y dependencias

N/A.

4.1.2.6. Requisitos pospuestos

N/A

4.1.3. Requisitos específicos

4.1.3.1 Interfaces externas

4.1.3.1.1 Interfaces de usuario

IU01: La aplicación se visualiza en una ventana redimensionable por el usuario con un tamaño mínimo de 600 x 800 pixels.

IU02: La aplicación posee una ventana de *registro*, figura 19, en la que se muestran dos campos de entrada, nombre y contraseña. A su vez en la parte inferior se visualizan dos botones: “entrar” y “crear cuenta”.

Figura 19. IU de registro

IU03: La interfaz *crear cuenta* muestra un formulario con los campos usuario, contraseña e email. Todos ellos son del tipo textBox.

En la parte inferior aparece un botón con el nombre “crear” y otro con el nombre “cancelar”. Al hacer click sobre el botón registro, deberá aparecer una ventana con el resultado de la operación.

IU04: La aplicación tiene una interfaz, *bibliotecas*, que muestra todas las bibliotecas creadas por el usuario, mostradas en lista y un *checkbox* al principio de cada una para poder elegir a cuál se desea acceder.

En la parte inferior se visualizan varios botones con los nombre “acceder”, “editar” , “eliminar” y por último, un botón con el nombre “crear biblioteca”.

IU05: En la interfaz *crear biblioteca*, se visualiza un formulario con los campos a introducir necesarios para crear una nueva biblioteca. Los campos del formulario son: nombre de la biblioteca del tipo textBox y descripción del tipo textArea. En la parte inferior se visualizan dos botones, “crear” y “cancelar”.

IU06: La interfaz *editar biblioteca*, muestra un formulario con los datos de la biblioteca seleccionada, dejando editar cada uno de ellos, el campo nombre de la biblioteca, es del tipo textBox. Se visualiza también un textArea para el campo descripción.

En la parte inferior se encuentran dos botones con los nombres, “editar” y ”cancelar”.

IU07: Tras realizar el registro y la selección de la biblioteca, se accede a la interfaz *miBiblioteca*.

En la parte superior se muestra centrado el siguiente texto: Biblioteca “*nombre de la biblioteca”*. A continuación, se visualiza un menú horizontal con las siguientes opciones: Inicio, buscar, crear, todos, perfil y un botón para desconectarse de la aplicación, con más detalle, en la figura 30.

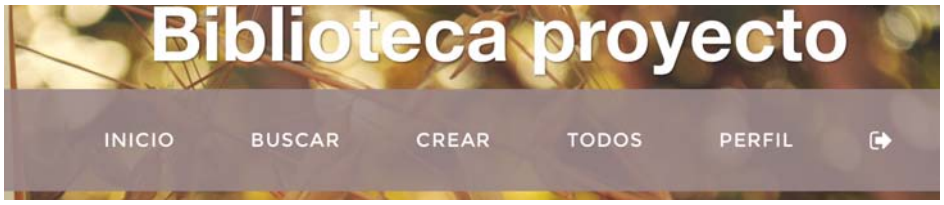


Figura 20. IU menú inicio

IU08: El menú horizontal muestra de manera más remarcada la opción seleccionada en ese momento, sirviendo de referencia de la operación que se está realizando.

IU09: "Inicio", opción del menú horizontal, muestra información relacionada con la última actividad de biblioteca elegida, mostrando los últimos libros añadidos. A su vez, muestra un botón "gestión de bibliotecas" para acceder a la interfaz bibliotecas.

IU10: "Buscar" muestra un entrada de texto, como se observa en la figura 21, donde se introduce el titulo del libro a buscar y un botón-imagen con un símbolo de buscar. A su vez, en la parte inferior se mostrará un nuevo botón denominado búsqueda avanzada, que permitirá realizar una búsqueda más precisa, proporcionando más campos de búsqueda relacionada con el libro a buscar.

Figura 21. IU búsqueda de un libro

IU11: Al presionar búsqueda avanzada, se muestra un nuevo formulario con los siguientes campos: título, autor, género e ISBN. En la parte inferior de la interfaz se muestra un botón-imagen con el símbolo de buscar.

IU12: Como resultado de la búsqueda, se muestra la carátula y el título del libro, como muestra la figura 22. En la parte inferior se muestran los siguientes botones, "editar" y "eliminar".

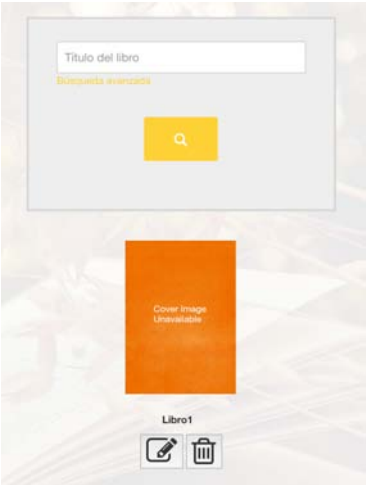


Figura 22. IU resultado búsqueda de un libro

IU13: Si se pasa el ratón por encima de la carátula, aparece un símbolo “más”, como el de la figura 23, que a su vez es un botón para acceder a toda la información relacionada con ese libro. Al presionar, se visualiza una ventana *pop up* con la información del libro.

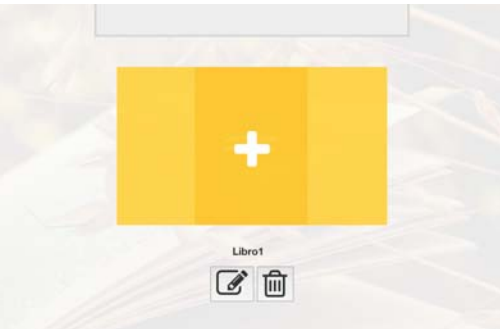


Figura 23. IU símbolo más en carátula

IU14: En la sección *crear*, se visualiza un formulario para introducir datos relacionados con un nuevo libro: título, autor, género, saga, editorial, páginas, ISBN, posición y resumen de libro. Además cuenta con un botón para poder subir una imagen del sistema. También se **visualiza un** botón con el nombre “crear”.

IU15: “Todos”, muestra una tabla dinámica con todos los libros que se han añadido a la biblioteca, como muestra la figura 24, facilitando la siguiente información: título autor, género y posición (fila y columna de la biblioteca física donde se encuentra).

Figura 24. IU tabla todos los libros

IU16: En todos los *textBox* está escrito el campo al que corresponde, al introducir texto en él, esa información desaparece dejando solo el texto introducido por el usuario. Al hacer click en un *textBox*, éste se remarca de color azul. Estas características se ven reflejadas en la figura 25.

Figura 25. IU campos formularios

IU17: Tras hacer click en editar un libro, se muestra la interfaz para editar un libro, donde se visualiza toda la información asociada al libro seleccionado, permitiendo editar cualquiera de los campos: título, autor, género, saga, editorial, páginas, ISBN, posición y resumen de libro. Además de poder subir una nueva carátula al sistema. En la parte inferior se muestran dos botones, editar y cancelar.

IU18: La interfaz "perfil" se visualiza una nueva interfaz con un formulario mostrando la información del usuario registrado con la posibilidad de editar todos los datos: nombre, contraseña y correo electrónico. A su vez en la parte inferior se muestran dos botones: "editar" y "eliminar usuario".

4.1.3.1.2. Interfaces hardware

N/A

4.1.3.1.3. Interfaces software

IU02: accesoAplicacion(String nombre, String password)
IU18: eliminarUsuario(int nombre)
IU03: nuevoUsuario(String nombre, String contraseña, String correo)
IU04: mostrarBibliotecas(String propietario)
IU05: crearBiblioteca(String nombre, String propietario, String Descripción)
IU04: eliminarBiblioteca(int id)
IU06: ActualizarBiblioteca(int id, String nombre, String propietario, String Descripción)
IU15: todosLibros()
IU10: buscarLibro(String título, string autor, string ISBN, string género)
IU014: añadirLibro(String título, String resumen, String autor, String género, String saga, int páginas, String editorial, String ISBN, int posición String carátula)
IU17: actualizarLibro(String título, String resumen, String autor, String género, String saga, int páginas, String editorial, String ISBN, int posición String carátula)
IU12: eliminarLibro(int id)
IU18: visualizarPerfil(int id)
IU18: editarPerfil(int id, String nombre, String contraseña, String correo)
IU04: accederBiblioteca(int idBiblioteca)

4.1.3.1.4. Interfaces de comunicación

N/A

4.1.3.2. Características del sistema

4.1.3.2.1. Registrar Usuario

4.1.3.2.1.1. Introducción/ Objeto de la característica

La aplicación debe permitir enviar datos de información personal a través de un formulario, con el objetivo de crear un nuevo usuario en la aplicación.

4.1.3.2.1.2. Secuencia de estímulo/respuesta

1. El usuario hace click (pulsar y soltar el botón izquierdo del ratón) sobre el botón “crear cuenta” que se encuentra en la ventana *registro*
2. Se visualiza la ventana *crear cuenta* con un formulario de registro. El usuario introduce los datos del formulario: nombre, contraseña y correo electrónico
3. Una vez completado el formulario, el usuario hace click en el botón “crear”
4. Si se ha completado correctamente el formulario y se ha realizado el registro, se accede a la interfaz de bibliotecas, para acceder a una existente o crear una nueva
5. Si algún dato obligatorio del formulario no ha sido rellenado y se hace click en el botón registrar, el sistema muestra un mensaje emergente recordando los campos a rellenar

4.1.3.2.1.3. Requisitos funcionales asociados

RU01: Los elementos obligatorios del formulario del registro son los siguientes: nombre, contraseña e email, con las siguientes restricciones:

nombre: máximo de 50 caracteres del alfabeto
contraseña: cualquier secuencia alfanumérica
email: [az-AZ]+@[az-aZ]+.com

RU02: El email es único, no se puede permitir el registro de un mismo email más de una vez
RU03: La aplicación mostrará un mensaje de error si no se puede realizar el registro
RU04: Si no se puede conectar con el servidor, se redirecciona a una nueva página mostrando que ha existido un error en la conexión.

4.1.3.2.2. Acceso a la aplicación

4.1.3.2.2.1. Introducción/ Objeto de la característica

La aplicación debe permitir el acceso, de personas registradas, a sus bibliotecas personales.

4.1.3.2.2.2. Secuencia de estímulo/respuesta

1. El usuario introduce nombre y contraseña en los campos correspondientes de la ventana *registro*
2. El usuario hace click sobre el botón “Acceder”
3. Si el acceso es correcto, se accede a la interfaz de selección de biblioteca. En caso contrario, no se permite el acceso a la aplicación y el usuario debe introducir de nuevo los campos usuario y contraseña

4.1.3.2.2.3. Requisitos funcionales asociados

AA01: EL acceso se realiza mediante nombre y contraseña. Ambos campos son obligatorios por los que no se pueden dejar vacíos
AA02: La contraseña debe ir cifrada utilizando sha-256
AA03: La conexión cliente-servidor debe ser a través del protocolo HTTPS
AA04: La sesión debe expirar si se cierra el navegador
AA05: La sesión debe expirar si existe un período de inactividad de 15 minutos
AA06: La aplicación mostrará un mensaje de error si el acceso a la librería no ha sido posible

4.1.3.2.3. Selección de biblioteca

4.1.3.2.3.1. Introducción/ Objeto de la característica

La aplicación deberá mostrar todas las bibliotecas que posee un usuario y seleccionar una de ellas para su utilización.

4.1.3.2.3.2. Secuencia de estímulo/respuesta

1. El usuario accede a la interfaz bibliotecas haciendo click en “gestión de bibliotecas” de la sección inicio del menú principal
2. El sistema muestra la interfaz bibliotecas
3. El usuario selecciona una de las bibliotecas mostradas
4. El usuario hace click en el botón “acceder”
5. Se muestra accede a la ventana principal inicio de la biblioteca seleccionada

4.1.3.2.3.3. Requisitos funcionales asociados

SB01: La aplicación solo puede permitir seleccionar una biblioteca de las mostradas
SB02: En caso de error en el acceso a la biblioteca, la aplicación muestra un ventana de error, indicando el problema ocurrido

4.1.3.2.4. Crear una nueva biblioteca

4.1.3.2.4.1. Introducción/ Objeto de la característica

La aplicación debe permitir crear diferentes bibliotecas para un mismo usuario.

4.1.3.2.4.2. Secuencia de estímulo/respuesta

1. El usuario hace click en el botón “crear biblioteca” de la ventana *bibliotecas*, pudiendo acceder desde la opción inicio del menú principal y haciendo click en “gestión de bibliotecas”
2. Aplicación muestra un formulario para crear la biblioteca
3. El usuario introduce la información requerida en el formulario, nombre de la biblioteca y descripción para crear una nueva biblioteca
4. El usuario hace click en el botón “crear”
5. Aplicación muestra mensaje de información de la operación
6. Aplicación muestra de nuevo la interfaz *bibliotecas* mostrando la nueva biblioteca añadida

4.1.3.2.4.3. Requisitos funcionales asociados

CB01: El campo “nombre de la biblioteca” del formulario es obligatorio.
CB02: En caso de error por conexión al servidor o en la creación de la propia biblioteca, la aplicación muestra un mensaje de error
CB03: Si el usuario cancela la creación, el sistema redirecciona a la ventana bibliotecas

4.1.3.2.5. Eliminar biblioteca existente

4.1.3.2.5.1. Introducción/ Objeto de la característica

La aplicación debe permitir eliminar bibliotecas del sistema

4.1.3.2.5.2. Secuencia de estímulo/respuesta

1. El usuario hace click en “gestión de bibliotecas” del menú inicio de la ventana principal
2. El sistema muestra la ventana bibliotecas
3. El usuario selecciona una biblioteca de la ventana *bibliotecas*
4. Posteriormente, el usuario hace click en el botón “eliminar”
5. La aplicación muestra mensaje de confirmación de la operación
6. El usuario hace click en “aceptar”
7. Tras aceptar, la aplicación muestra un mensaje con el resultado de la operación
8. Se muestra de nuevo la ventana *bibliotecas* tras la operación

4.1.3.2.5.3. Requisitos funcionales asociados

EB01: No se permite seleccionar más de una biblioteca

EB02: Al tratarse de un proceso que no hay marcha atrás se debe mostrar un mensaje de confirmación al usuario con el siguiente texto: “La operación que se va a realizar no tiene marcha atrás, ¿desea continuar?”

4.1.3.2.6. Buscar un libro

4.1.3.2.6.1. Introducción/ Objeto de la característica

La aplicación debe permitir buscar cualquier libro de la biblioteca seleccionada con antelación

4.1.3.2.6.2. Secuencia de estímulo/respuesta

1. En el menú de la ventana *miBiblioteca*, el usuario hace click en “buscar”
2. La aplicación muestra un campo de texto para introducir el título y un botón de búsqueda
3. A la izquierda se muestra un botón “búsqueda avanzada” donde se permitirá realizar una búsqueda introduciendo alguno de los siguientes datos: título, autor o ISBN
4. En caso de no utilizar la búsqueda avanzada, el usuario solo introduce el nombre del libro a buscar

5. El usuario hace click sobre el botón con una imagen de lupa o el botón buscar (dependiendo de la búsqueda seleccionada) para proceder a la búsqueda
6. La aplicación muestra el resultado de la búsqueda con la portada del libro y en la parte inferior el título del mismo
7. El usuario hace click en la imagen del libro para visualizar la información del mismo
8. El sistema muestra una ventana del tipo *pop-up* mostrando la siguiente información del libro: título, sinopsis, autor, género, páginas, saga e ISBN

4.1.3.2.6.3. Requisitos funcionales asociados

BL01: No se permite buscar un campo vacío. El campo de nombre debe estar relleno
BL02: En caso de enviar un campo vacío, la aplicación muestra un mensaje de error indicando que se debe introducir algún valor en dicho campo
BL03: Si no se encuentra ningún libro, el sistema en mostrará por pantalla el siguiente texto: "ningún libro encontrado en la biblioteca"

4.1.3.2.7. Crear libro

4.1.3.2.7.1. Introducción/ Objeto de la característica

La aplicación debe permitir incorporar nuevos libros a la biblioteca seleccionada

4.1.3.2.7.2. Secuencia de estímulo/respuesta

1. En el menú de la ventana *miBiblioteca*, el usuario hace click en "crear"
2. La aplicación muestra el formulario con los campos necesarios para crear un nuevo libro
3. El usuario completa el formulario con los datos siguientes: título, autor, género, saga, editorial, páginas, ISBN, colocación, resumen e imagen para poder crear un nuevo libro
4. El usuario hace click en el botón "crear"
5. La aplicación muestra una ventana con el resultado de la operación
6. La aplicación redirecciona a la sección *inicio* de la ventana *miBiblioteca*

4.1.3.2.7.3. Requisitos funcionales asociados

CL01: El campo ISBN será único
CL02: El campo colocación consta de dos campos, fila y columna
CL03: Los campos título, autor, género y editorial, son obligatorios
CL04: La aplicación muestra un mensaje de error en caso de no realizarse la creación del libro de manera satisfactoria

4.1.3.2.8. Editar información de un libro

4.1.3.2.8.1. Introducción/ Objeto de la característica

La aplicación debe permitir modificar toda la información de un libro que ya ha sido creado

4.1.3.2.8.2. Secuencia de estímulo/respuesta

1. El usuario hace click en el botón buscar del menú de la ventana *miBiblioteca*
2. Aplicación muestra formulario de búsqueda
3. Usuario realiza una búsqueda simple o avanzada
4. Aplicación muestra los resultados de la búsqueda
5. El usuario hace click en el botón "editar" del libro deseado
6. Aplicación un formulario con la información actual de ese libro
7. El usuario modifica los datos deseados
8. El usuario hace click en el botón "editar" de la interfaz de editado
9. La aplicación muestra una ventana de confirmación para poder continuar con el editado
10. La aplicación muestra una ventana confirmando los cambios
11. Aplicación redirecciona a la ventana principal de la biblioteca

4.1.3.2.8.3. Requisitos funcionales asociados

EL01: Los campos título, autor, género y editorial, son obligatorios
EL02: La aplicación muestra un mensaje de error en caso de no realizarse la creación del libro de manera satisfactoria

4.1.3.2.9. Editar información de una biblioteca

4.1.3.2.9.1. Introducción/ Objeto de la característica

La aplicación debe permitir cambiar la información referida a una biblioteca ya creada

4.1.3.2.9.2. Secuencia de estímulo/respuesta

1. El usuario hace click en "inicio" del menú principal y a su vez hace click en el botón "gestión de bibliotecas"
2. El sistema muestra la ventana *bibliotecas* con todas las bibliotecas pertenecientes al usuario registrado
3. El usuario selecciona en el *checkbox* la biblioteca que desee. Hace click en el botón editar
4. Se visualiza un formulario con la información actual de la biblioteca seleccionada, nombre y descripción
5. El usuario realiza alguna modificación de los campos nombre y descripción, hace click en el botón "editar"
6. La aplicación muestra un mensaje de confirmación
7. La aplicación visualiza una ventana confirmando el cambio realizado
8. La aplicación redirecciona a la ventana *bibliotecas*, mostrando todas la bibliotecas que tiene el usuario y el cambio realizado

4.1.3.2.9.3. Requisitos funcionales asociados

EIB01: No se puede seleccionar más de una biblioteca al mismo tiempo para realizar algún cambio
EIB02: El sistema mostrará un mensaje de error en caso de haberse producido correctamente el proceso requerido
EIB03: El usuario puede volver atrás sin realizar cambios haciendo click en el botón “cancelar”

4.1.3.2.10. Visualizar todos los libros

4.1.3.2.10.1. Introducción/ Objeto de la característica

La aplicación debe mostrar una tabla con todos los libros que posee la biblioteca

4.1.3.2.10.2. Secuencia de estímulo/respuesta

1. Usuario hace click en el botón “todos” del menú de la ventana *miBiblioteca*
2. Se visualiza una tabla con todos los libros de la biblioteca, que serán mostrados en orden alfabético. Para cada libro, se muestra la siguiente información: título, autor, género, fila, columna y leído

4.1.3.2.10.3. Requisitos funcionales asociados

VL01: el contenido de la tabla debe poder ordenarse por cualquier campo de la misma
VL02: Se debe poder buscar por cualquier caracter que se encuentre en la tabla

4.1.3.2.11. Visualizar/editar perfil

4.1.3.2.11.1. Introducción/ Objeto de la característica

La aplicación debe permitir acceder a la información de la cuenta del usuario y su modificación

4.1.3.2.11.2. Secuencia de estímulo/respuesta

1. El usuario hace click en el botón “perfil”, que se encuentra en la esquina superior derecha de la ventana *miBiblioteca*
2. Se muestra un formulario con los datos actuales del usuario registrado
3. El usuario modifica los campos del formulario: nombre, contraseña y correo electrónico
4. El usuario hace click en el botón “editar”
5. La aplicación muestra un mensaje de confirmación
6. Usuario acepta el editado
7. La aplicación recarga la interfaz de gestión del perfil

4.1.3.2.11.3. Requisitos funcionales asociados

VP01: Los elementos obligatorios del formulario del perfil son los siguientes: nombre, contraseña e email, con las siguientes restricciones:

nombre: máximo de 50 caracteres del alfabeto
contraseña: cualquier secuencia alfanumérica
email: [az-AZ]+@[az-aZ]+.com

VP02: El campo email es único, no se puede editar en caso de que un email ya esté en uso
VP03: La aplicación mostrará un mensaje de error si no se puede editar la información del perfil

4.1.3.2.12. Eliminar perfil

4.1.3.2.12.1. Introducción/ Objeto de la característica

La aplicación debe permitir eliminar una cuenta de usuario

4.1.3.2.12.2. Secuencia de estímulo/respuesta

1. El usuario hace click en el botón “perfil”, que se encuentra en la esquina superior derecha de la ventana *miBiblioteca*
2. Se muestra la interfaz de gestión del perfil, mostrando los datos del usuario conectado
3. El usuario hace click en el botón “eliminar usuario”
4. La aplicación muestra un mensaje de confirmación
5. Usuario confirma la eliminación de su cuenta
6. Aplicación elimina toda la información relacionada con el usuario y redirecciona a la página de login

4.1.3.2.12.3. Requisitos funcionales asociados

EP01: En caso de error, el sistema muestra un mensaje de error y vuelve a la interfaz perfil
EP02: Tras eliminar el usuario, el sistema elimina todos los datos relacionados con el mismo. Todas las bibliotecas y libros de las mismas

4.1.3.2.13. Cerrar sesión (*log off*)

4.1.3.2.13.1. Introducción/ Objeto de la característica

La aplicación debe permitir salir de la aplicación y cerrar nuestra cuenta de usuario en cualquier momento

4.1.3.2.13.2. Secuencia de estímulo/respuesta

1. El usuario hace click en el botón “desconectar” de la parte superior derecha de cualquier ventana de la aplicación

2. La aplicación cierra la sesión y redirecciona a la ventana *registro*, para introducir de nuevo las credenciales

4.1.3.2.13.3. Requisitos funcionales asociados

CS01: La aplicación debe eliminar las cookies de sesión creadas

4.1.3.2.14. Eliminar libro

4.1.3.2.14.1. Introducción/ Objeto de la característica

La aplicación debe permitir eliminar un libro de una biblioteca previamente seleccionada

4.1.3.2.14.2. Secuencia de estímulo/respuesta

1. El usuario hace click en el botón “buscar” del menú principal de la aplicación
2. Aplicación muestra formulario de búsqueda
3. El usuario realiza la búsqueda de un libro
4. La aplicación muestra el resultado de la búsqueda
5. El usuario hace click en el botón “eliminar”
6. La aplicación muestra un mensaje de confirmación
7. El usuario confirma el mensaje y se redirecciona a la ventana inicio de la aplicación

4.1.3.2.14.3. Requisitos funcionales asociados

ELI1: En caso de no poder eliminar el libro deseado, el sistema mostrará el mensaje de error correspondiente

4.1.3.3. Requisitos de rendimiento

N/A

4.1.3.4. Restricciones de diseño

La notación a utilizar será UML y la metodología de desarrollo será el Proceso Unificado

4.1.3.5. Atributos software del sistema

4.1.3.5.1. Fiabilidad

La aplicación debe gestionar los posibles errores en tiempo de ejecución de tal forma que siempre sea posible volver a un estado consistente o finalizar la aplicación de forma elegante

4.1.3.5.2. Disponibilidad

La aplicación estará disponible de manera continua de lunes a sábado.

Los Domingos de 00:00-03:00 AM se detiene el servicio para el mantenimiento del mismo.

Se realiza un *back-up* automático completo del servicio todos los días de 02:00 a 03:00 AM, el cual debe ser realizado permitiendo el funcionamiento de la aplicación sin ningún problema.

4.1.3.5.3. Seguridad

Para acceder, previamente se debe estar autenticado, la clave deberá estar cifrada mediante sha-256, y la conexión se deberá realizar mediante HTTPS

4.1.3.5.4. Facilidad de mantenimiento

La aplicación debe ser fácilmente extensible

4.1.3.5.5. Portabilidad

La aplicación debe proporcionar todas las funciones, para todos los navegadores Web existentes.

4.2. Casos de uso y diagramas de actividades

En este apartado se muestran los casos de uso en la figura 26 y a continuación, en las siguientes tablas e imágenes, los diagramas de caso de uso en formato extendido y los diagramas de actividades desarrollados para esta aplicación.

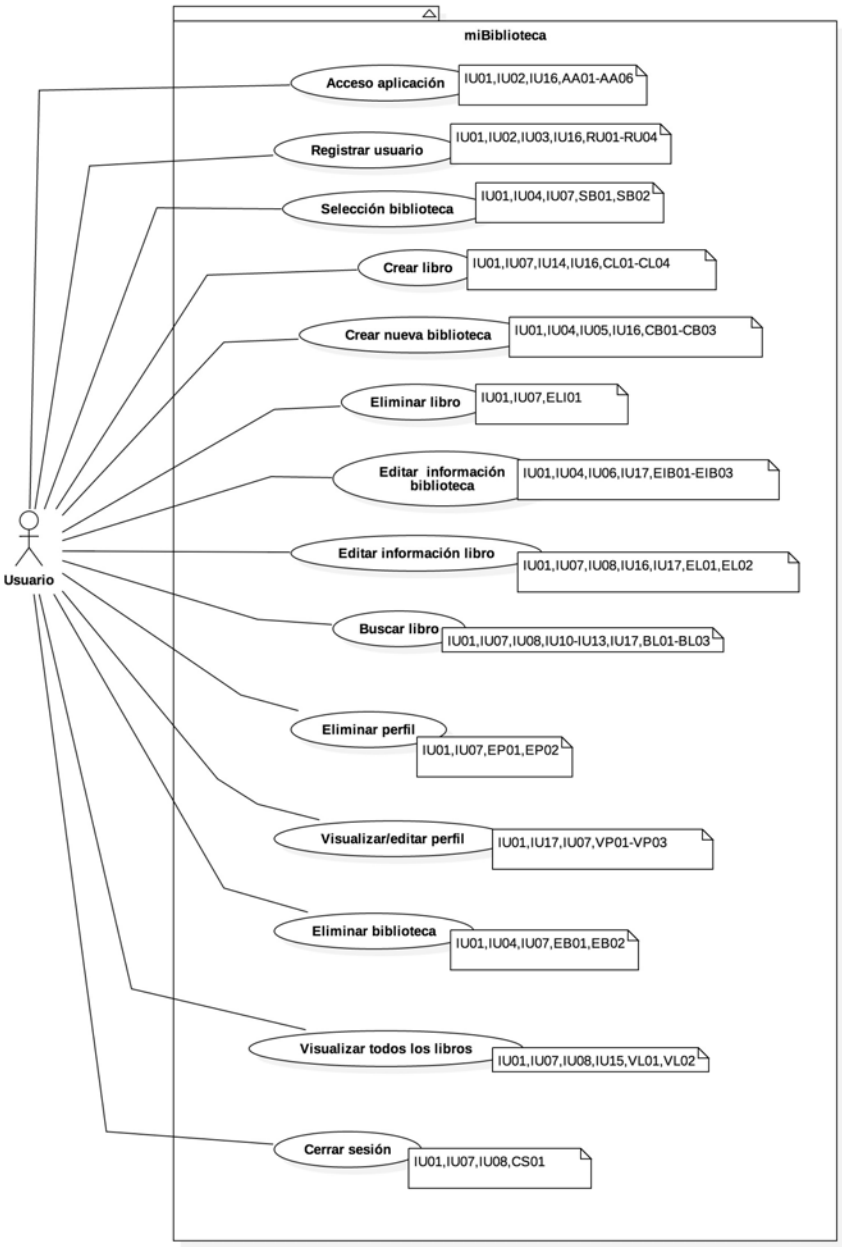


Figura 26. Diagrama de casos de uso

4.2.1. Caso de uso extendido y diagrama de actividad: Registrar usuario

Caso de Uso	Registrar usuario
Descripción:	Usuario crea una nueva cuenta
Actores:	Usuario
Precondiciones:	El usuario no debe estar registrado
Flujo Normal	
Usuario	Sistema
01-Pulsar “crear cuenta”	
	02-Mostrar formulario de nuevo registro
03-Rellenar campos: nombre y contraseña del formulario - “Nombre usuario” -“Correo electrónico” -“Contraseña”	
04-pulsar “crear”	
	05-Validar entradas
	06-Crea la cuenta del usuario
	07-Hace login al usuario y carga interfaz bibliotecas
Poscondiciones:	
Flujo Alternativo	
Usuario	Sistema
	05-Validar entradas
	06-Mostrar mensaje de error por la validación de la entrada
	07-Volver al paso 03 del flujo principal

Usuario	Sistema
	02-Mostrar formulario de nuevo registro
03-Pulsar “cancelar”	
	04-Carga pantalla de login

Tabla 8. Caso de uso extendido: Registrar usuario

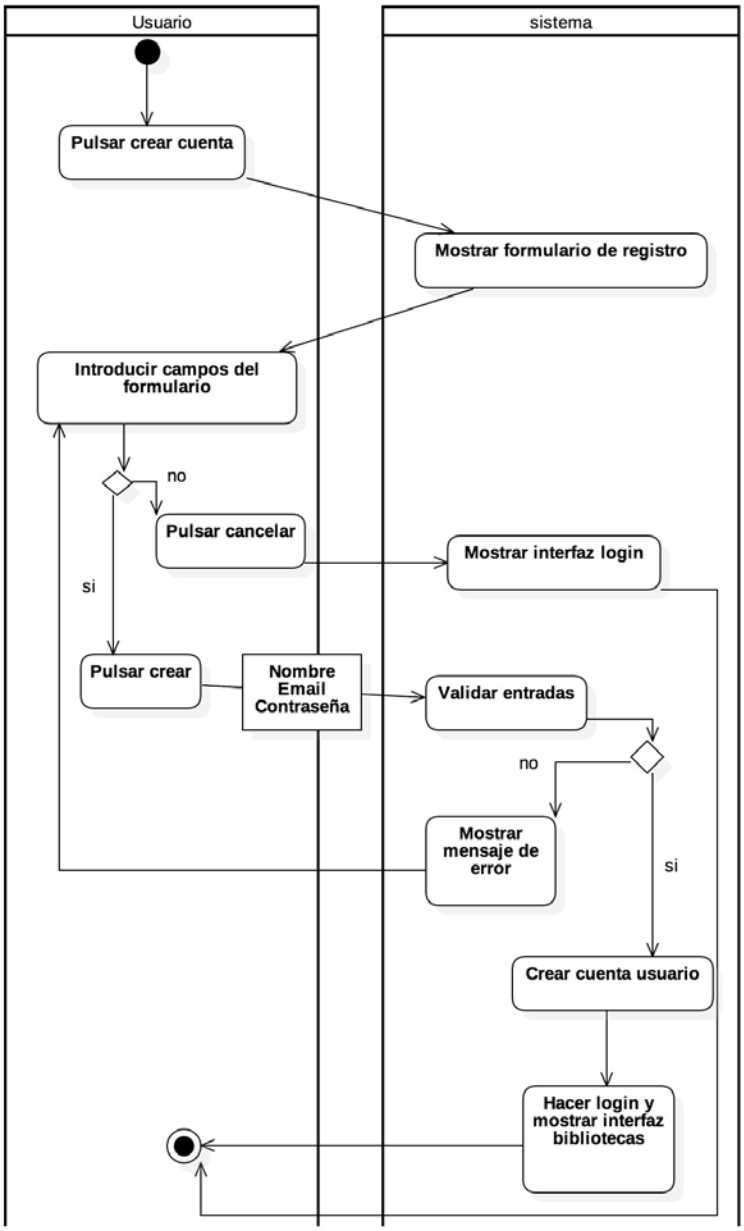


Figura 27. Diagrama de actividad: Registrar usuario

4.2.2. Caso de uso extendido y diagrama de actividad: Acceso aplicación

Caso de Uso	Acceso aplicación
Descripción:	Usuario hace login para entrar en la aplicación
Actores:	Usuario
Precondiciones:	El usuario debe tener una cuenta creada con anterioridad. Se visualiza una ventana para rellenar los campos de nombre y contraseña
Flujo Normal	
Usuario	Sistema
01-Rellena datos del formulario: “nombre” y “contraseña”	
02-pulsar “Acceder”	
	03-Validar formulario
	04-Carga pantalla selección de biblioteca
Poscondiciones:	
Flujo Alternativo	
Usuario	Sistema
	03-Validar formulario
	04-Mostrar mensaje de error por validación de las entradas
	05-Volver al paso 01 del flujo principal

Tabla 9. Caso de uso extendido: Acceso aplicación

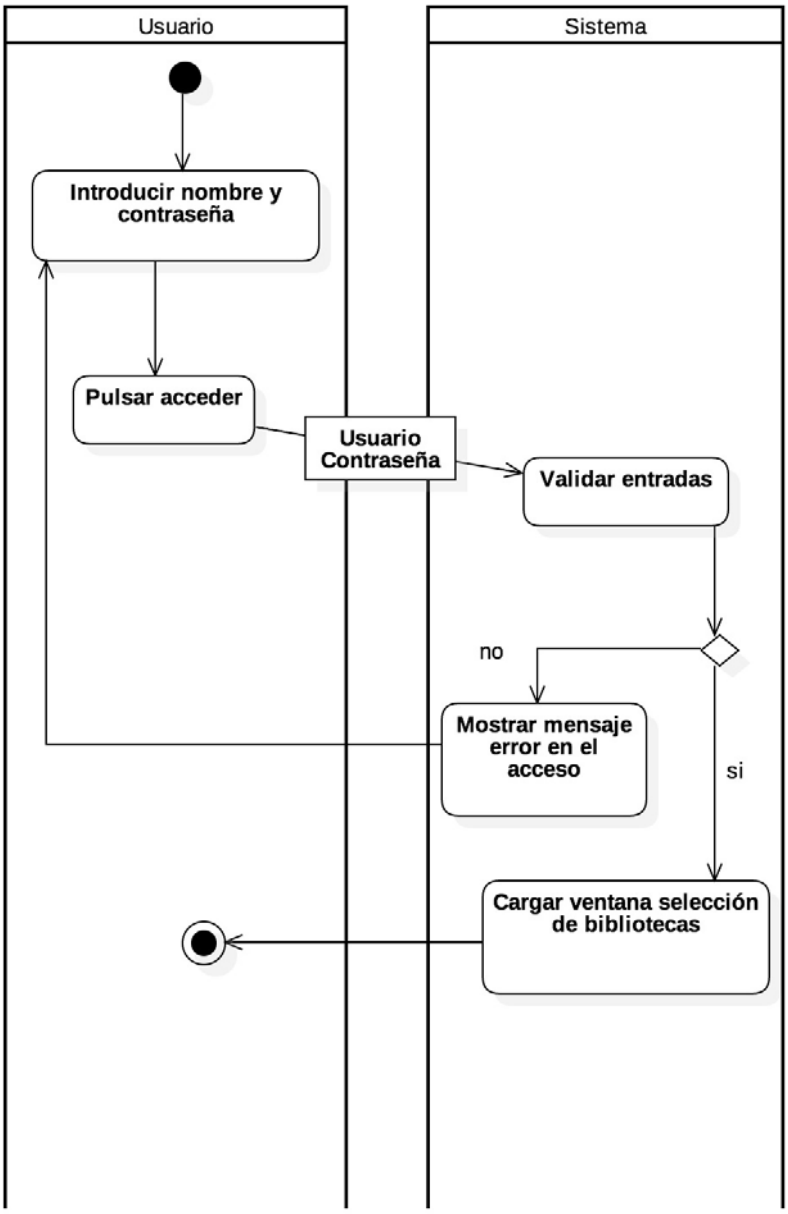


Figura 28. Diagrama de actividad: Acceso aplicación

4.2.3. Caso de uso extendido y diagrama de actividad: Crear nueva biblioteca

Caso de Uso	Crear nueva biblioteca
Descripción:	Usuario crea una nueva biblioteca
Actores:	Usuario
Precondiciones:	El usuario debe haber hecho login. Se visualiza la sección inicio, opción del menú horizontal de la ventana miBiblioteca
Flujo Normal	
Usuario	Sistema
01-Hacer click en “gestión de bibliotecas”	
	02-Muestra ventana bibliotecas
03-Hacer click en “crear biblioteca”	
	04-Mostrar formulario
05-Introducir los datos obligatorios del formulario: “nombre” y “descripción”	
06-Pulsar “crear” para enviar formulario	
	07-Validar formulario
	08-Crea la biblioteca
	09-Mostrar mensaje “biblioteca creada correctamente”
	10-Carga página “bibliotecas” para seleccionar biblioteca deseada
Poscondiciones:	
Flujo Alternativo	
Usuario	Sistema

	07-Validar entradas
	08-Mostrar mensaje de error por la validación de la entrada
	09-Volver al paso 05 del flujo principal
Usuario	Sistema
	04-Mostrar formulario
05-Pulsar “cancelar”	
	06-Carga pantalla bibliotecas

Tabla 10. Caso de uso extendido: Crear nueva biblioteca

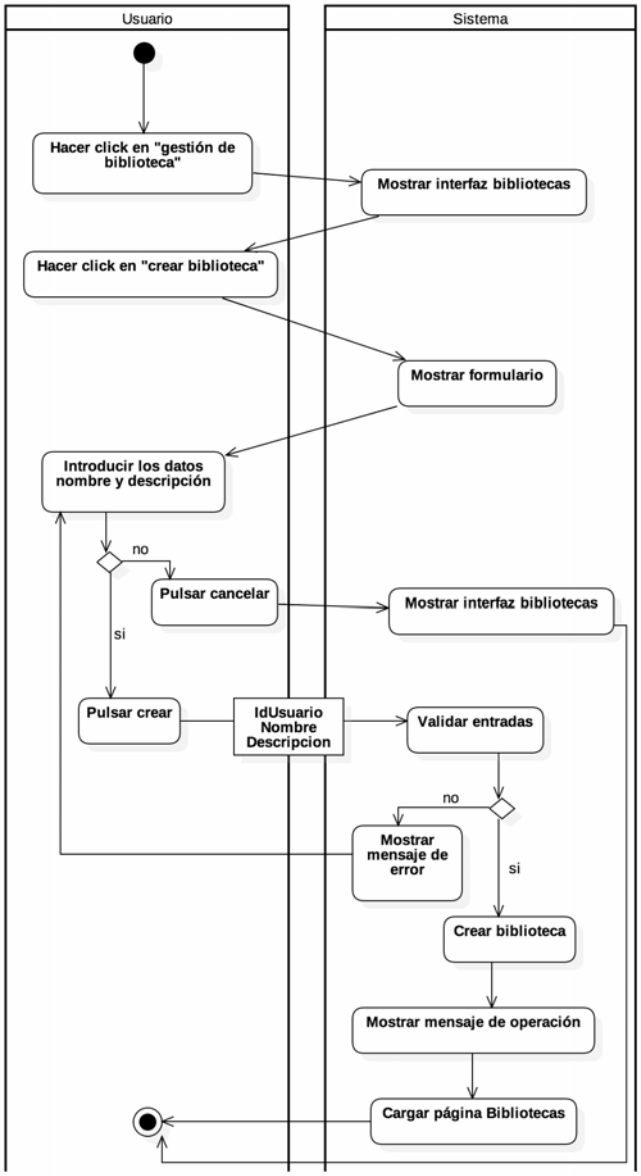


Figura 29. Diagrama de actividad: Crear nueva biblioteca

4.2.4. Caso de uso extendido y diagrama de actividad: Selección biblioteca

Caso de Uso	Selección biblioteca
Descripción:	Usuario elige una biblioteca determinada
Actores:	Usuario
Precondiciones:	El usuario debe haber creado una o más bibliotecas y haber hecho login. Se visualiza la ventana principal miBiblioteca
Flujo Normal	
Usuario	Sistema
01-Pulsar “gestión de biblioteca” del menú “inicio”	
	02-Muestra interfaz bibliotecas
03-Seleccionar una biblioteca de la lista mostrada	
04-Pulsar “Acceder”	
	05-Validar entrada
	06-Cargar pantalla de inicio de la nueva biblioteca seleccionada
Poscondiciones:	
Flujo Alternativo	
Usuario	Sistema
	05-Validar entrada
	06-Mostrar mensaje de error de acceso
	07-Volver automáticamente al paso 03 del flujo principal

Tabla 11. Caso de uso extendido: Selección biblioteca

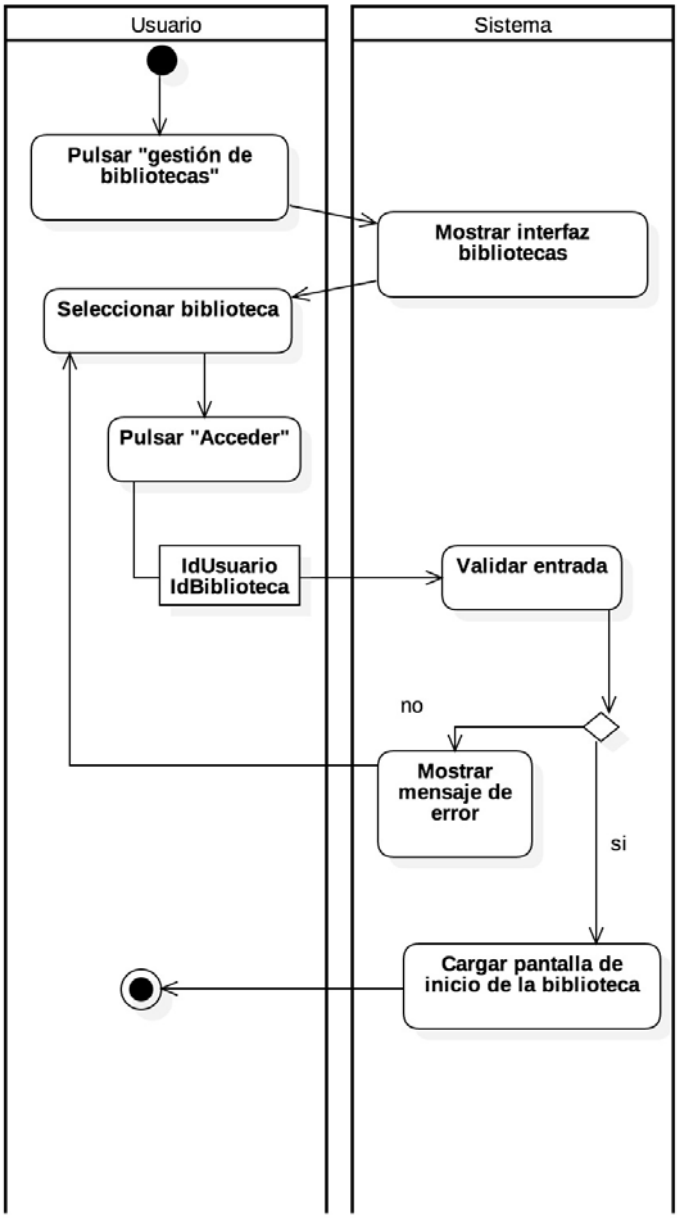


Figura 30. Diagrama de actividad: Selección biblioteca

4.2.5. Caso de uso extendido y diagrama de actividad: Eliminar biblioteca

Caso de Uso	**Eliminar biblioteca**
Descripción:	Usuario elimina una biblioteca creada
Actores:	Usuario
Precondiciones:	El usuario debe haber creado una o más bibliotecas y haber hecho login. Se visualiza la ventana principal miBiblioteca
Flujo Normal	
Usuario	Sistema
01-Pulsar “gestión de biblioteca” del menú “inicio”	
	02-Muestra interfaz bibliotecas
03-Seleccionar biblioteca	
04-Pulsar “eliminar”	
	05-Mensaje de confirmación
06-Confirmar eliminar	
	07-Elimina biblioteca seleccionada
	08-Mensaje de resultado y carga ventana bibliotecas
Flujo Alternativo	
Usuario	Sistema
	05-Mensaje de confirmación
06-Cancelar mensaje confirmación	
	07-Volver al paso 03 del flujo principal

Tabla 12. Caso de uso extendido: Eliminar biblioteca

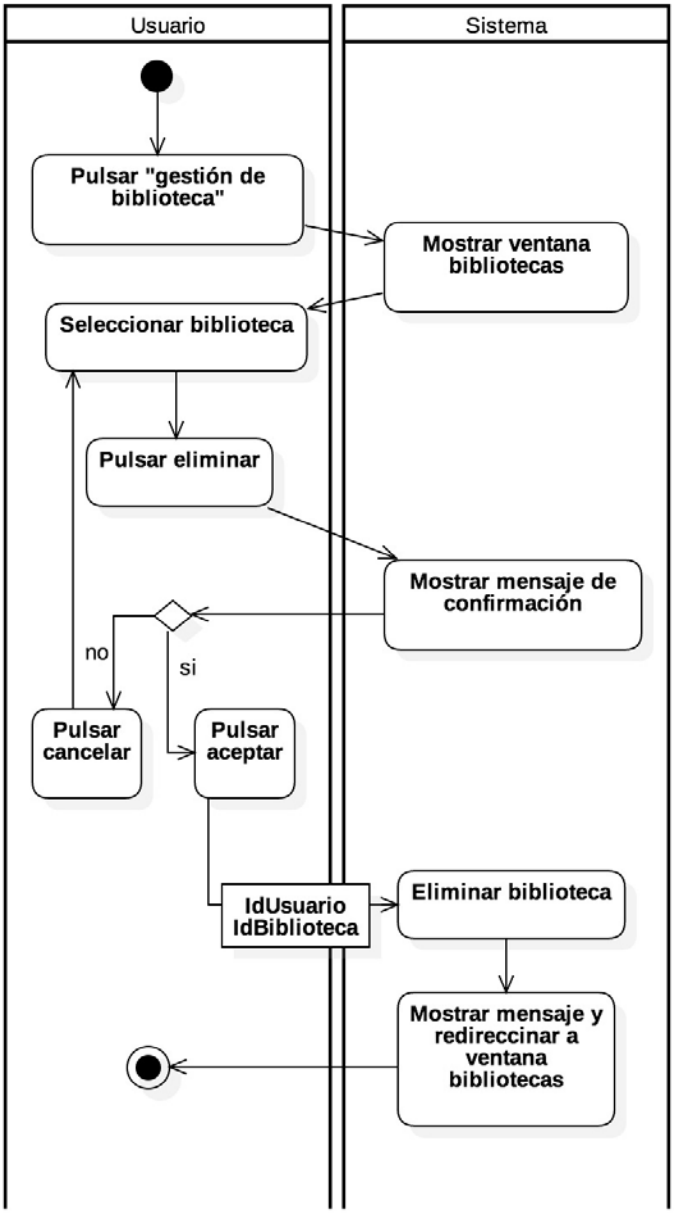


Figura 31. Diagrama de actividad: Eliminar biblioteca

4.2.6. Caso de uso extendido y diagrama de actividad: Buscar libro

Caso de Uso	Buscar libro
Descripción:	Usuario puede buscar cualquier libro de la biblioteca
Actores:	Usuario
Precondiciones:	El usuario debe haber hecho login y haber seleccionado una biblioteca. Se visualiza la ventana principal de la aplicación, miBiblioteca
Flujo Normal	
Usuario	Sistema
01-Pulsar “buscar” del menú principal	
	02-Muestra formulario de búsqueda
03-Introducir “título” en el campo del formulario	
04-Pulsar botón de búsqueda	
	05-Validar formulario
	06-Muestra portada del libro encontrado y su título
07-Hace click en la carátula del libro	
	08-Muestra ventana pop-up con la siguiente información del libro: título, sinopsis, autor, género, páginas, saga e ISBN
Poscondiciones:	
Flujo Alternativo	
Usuario	Sistema
	05-Validar formulario

	06-Mensaje en pantalla:“ningún libro encontrado en la biblioteca”
	07-Volver al paso 03 del flujo principal
Usuario	Sistema
	02-Muestra formulario de búsqueda
03-pulsar “búsqueda avanzada”	
	04-Muestra formulario búsqueda avanzada
05-Introducir datos del formulario: “título”,”autor”,”ISBN” y ”genero”	
06-Hace click en botón “buscar”	
	07-Volver al paso 04 del flujo principal

Tabla 13. Caso de uso extendido: Buscar libro

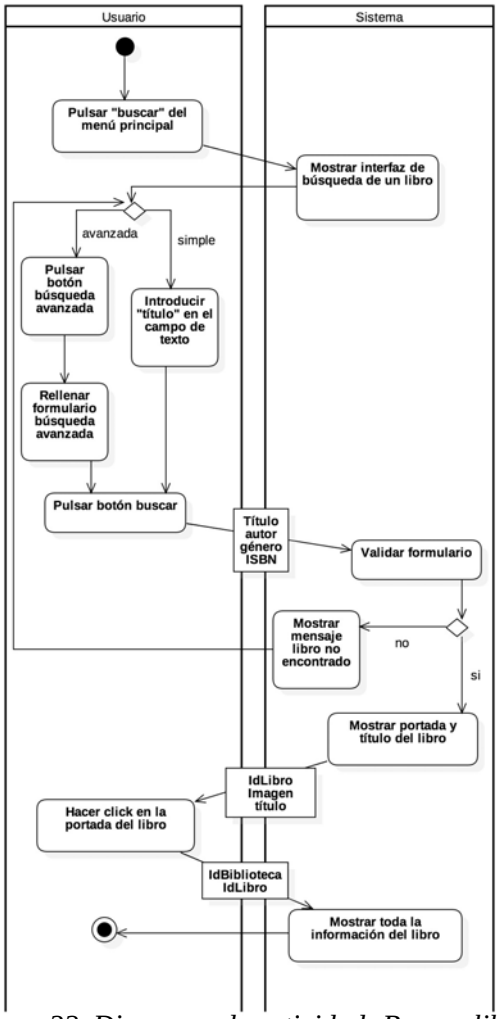


Figura 32. Diagrama de actividad: Buscar libro

4.2.7. Caso de uso extendido y diagrama de actividad: Editar información biblioteca

Caso de Uso	Editar información biblioteca
Descripción:	Usuario edita la información que identifica a una biblioteca
Actores:	Usuario
Precondiciones:	El usuario debe haber hecho login y seleccionado una biblioteca. Se visualiza la venta principal de la aplicación, miBiblioteca
Flujo Normal	
Usuario	Sistema
01-Pulsar “gestión de bibliotecas” del menú “inicio”	
	02-Muestra ventana bibliotecas
03-Seleccionar biblioteca	
04-Pulsar “editar”	
	05-Muestra formulario de editado
06-Introducir los datos del formulario: “nombre” y “descripción”	
07-Pulsa “editar”	
	08-Mensaje de confirmación
09-Confirmar editado	
	10-Validar entradas
	11-Modifica información de la biblioteca
	12-Cargar pantalla confirmando el editado
	13-Redirecciona a la ventana bibliotecas
Poscondiciones:	

Flujo Alternativo	
Usuario	Sistema
	10-Validar entradas
	11-Mostrar mensaje de error por la validación de la entrada
12-Cerrar mensaje de error	
	13-Volver al paso 06 del flujo principal
Usuario	Sistema
	08-Mensaje de confirmación
09-Cancelar mensaje confirmación	
	10-Volver al paso 03 del flujo principal
Usuario	Sistema
07-pulsa “cancelar”	
	08-Volver al paso 03 del flujo principal

Tabla 14. Caso de uso extendido: Editar información biblioteca

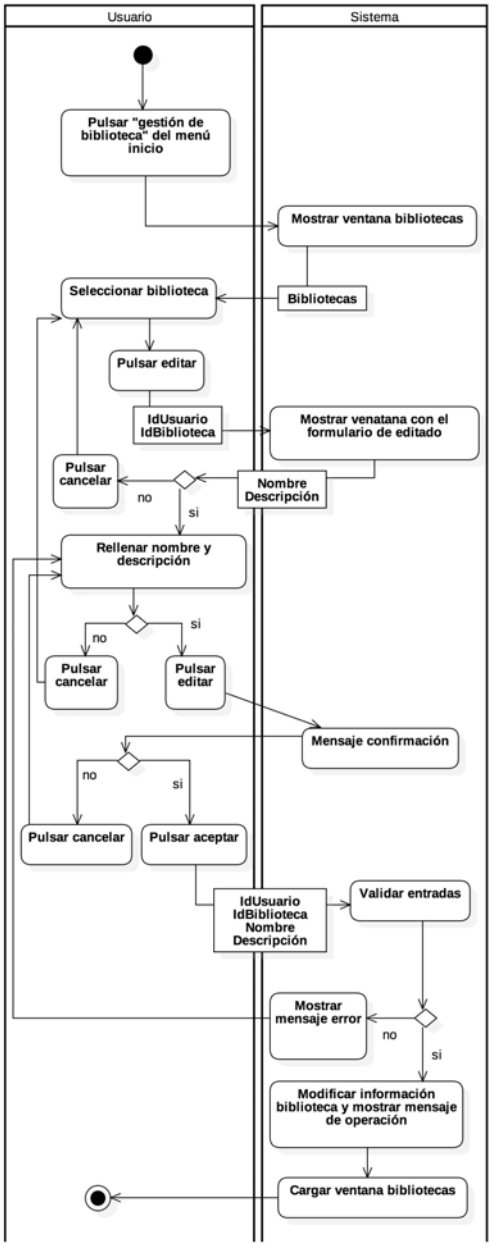


Figura 33. Diagrama de actividad: Editar información biblioteca

4.2.8. Caso de uso extendido y diagrama de actividad: Editar información de un libro

Caso de Uso	Editar información de un libro
Descripción:	Usuario edita la información de un libro de la biblioteca
Actores:	Usuario
Precondiciones:	El usuario debe haber hecho login y seleccionado una biblioteca. Se visualiza la venta principal de la aplicación, miBiblioteca
Flujo Normal	
Usuario	Sistema
01-Pulsar “buscar” del menú principal	
02-Realizar búsqueda de un libro	
	03-Muestra resultado de la búsqueda
04-Pulsar “editar” del libro deseado	
	05-Muestra formulario rellenado con los datos relacionados con ese libro
06-Modifica los datos del formulario: ”título”, “resumen”, “autor”, “género”, “saga”, “páginas”, “editorial”, “ISBN”, “posición” y “carátula”	
07-Enviar formulario pulsando “editar”	
	08-Mensaje de confirmación
09-Confirmar editado	
	10-Validar entradas
	11-Modifica la información del libro
	12-Cargar pantalla confirmando el editado
	13-Carga ventana “inicio” de la interfaz miBiblioteca
Poscondiciones:	

Flujo Alternativo	
Usuario	Sistema
	11-Mostrar mensaje de error por la validación de la entrada
12-Cerrar mensaje de error	
	13-Volver al paso 06 del flujo principal
Usuario	Sistema
	08-Mensaje de confirmación
09-Cancelar mensaje confirmación	
	10-Volver al paso 02 del flujo principal

Tabla 15. Caso de uso extendido: Editar información de un libro

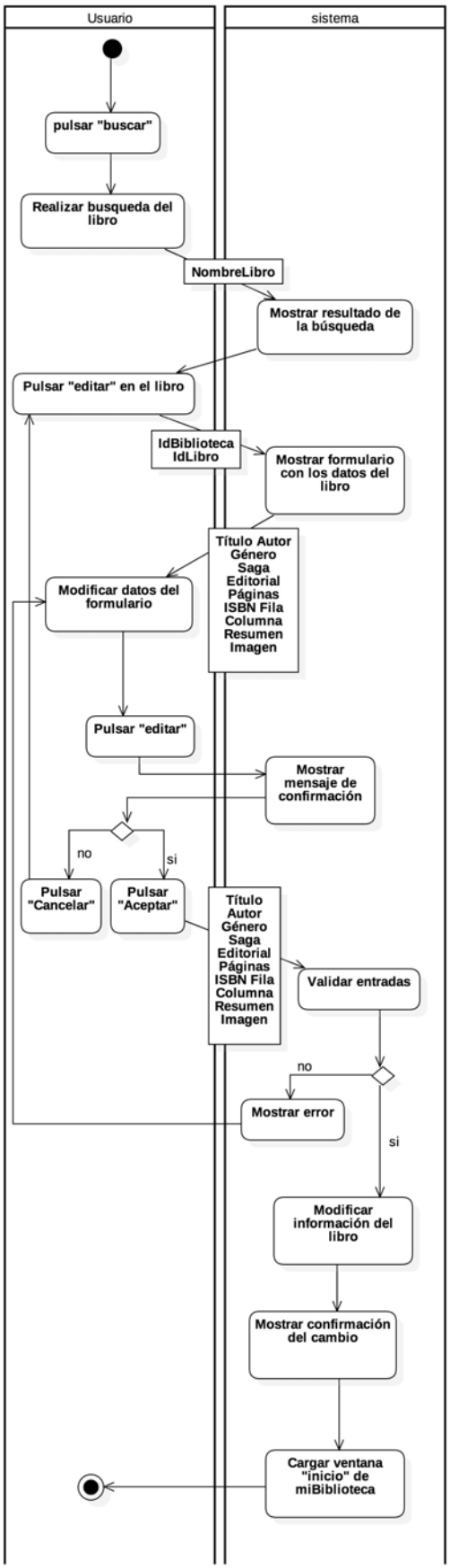


Figura 34. Diagrama de actividad: Editar información de un libro

4.2.9. Caso de uso extendido y diagrama de actividad: Crear libro

Caso de Uso	Crear libro
Descripción:	Usuario crea un nuevo libro
Actores:	Usuario
Precondiciones:	El usuario debe haber hecho login y haber seleccionado una biblioteca. Se visualiza la venta principal de la aplicación, miBiblioteca
Flujo Normal	
Usuario	Sistema
01-Pulsar “crear” del menú de la interfaz miBiblioteca	
	02-Muestra un formulario de creación
03-Introducir los datos obligatorios del formulario: “título”, “autor”, “editorial” e “ISBN”	
04-Pulsar “crear” para enviar el formulario	
	05-Validar entradas
	06-Crea el libro
	07-Cargar página indicando el resultado de la operación
	08-Cargar página de inicio de la biblioteca
Poscondiciones:	
Flujo Alternativo	
Usuario	Sistema
	06-Mostrar mensaje de error por la validación de la entrada
	07-volver al paso 03 del flujo principal

Tabla 16. Caso de uso extendido: Crear libro

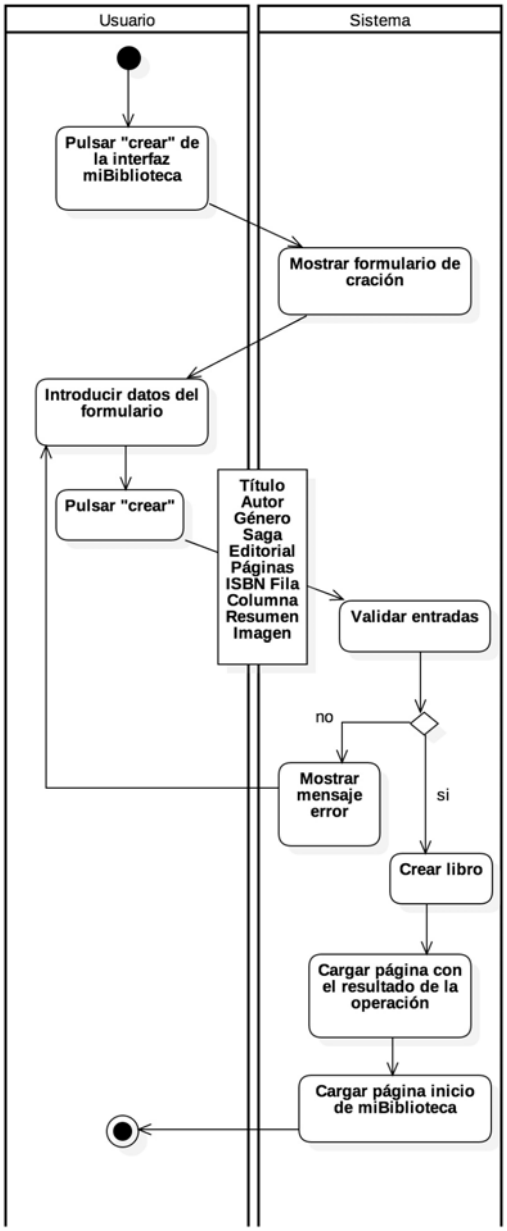


Figura 35. Diagrama de actividad: Crear libro

4.2.10. Caso de uso extendido y diagrama de actividad: Visualizar todos los libros

Caso de Uso	Visualizar todos los libros
Descripción:	Usuario visualiza una lista con todos los libros de la biblioteca
Actores:	Usuario
Precondiciones:	El usuario debe estar registrado y haber seleccionado una biblioteca. Se visualiza la venta principal de la aplicación, miBiblioteca
Flujo Normal	
Usuario	Sistema
01-Pulsar “todos” del menú de la interfaz miBiblioteca	
	02-Muestra en una tabla todos los libros de la biblioteca seleccionada con la siguiente información: “título”, “autor”, “género”, “fila”, “columna” y “leído”
Poscondiciones:	
Flujo Alternativo	

Tabla 17. Caso de uso extendido: Visualizar todos los libros

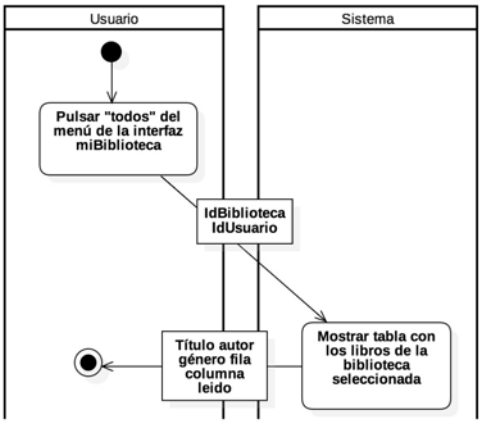


Figura 36. Diagrama de actividad: Visualizar todos los libros

4.2.11. Caso de uso extendido y diagrama de actividad: Cerrar sesión

Caso de Uso	Cerrar sesión
Descripción:	El usuario puede cerrar la sesión cuando quiera
Actores:	Usuario
Precondiciones:	El usuario ha de estar registrado. Se visualiza cualquier ventana de la aplicación
Flujo Normal	
Usuario	Sistema
01-Pulsar “desconectar” en la parte superior derecha de la ventana	
	02-Cierra sesión del usuario
	03-Redirección a la página de login
Poscondiciones:	
Flujo Alternativo	

Tabla 18. Caso de uso extendido: Cerrar sesión

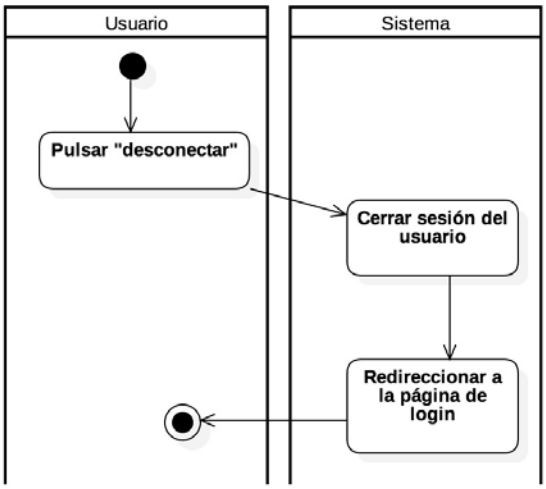


Figura 37. Diagrama de actividad: Cerrar sesión

4.2.12. Caso de uso extendido y diagrama de actividad: Eliminar perfil

Caso de Uso	Eliminar perfil
Descripción:	Usuario elimina su cuenta del sistema
Actores:	Usuario
Precondiciones:	El usuario debe haber echo login. Se visualiza cualquier ventana de la aplicación
Flujo Normal	
Usuario	Sistema
01-Pulsar “perfil” en la esquina superior derecha	
	02-Muestra interfaz de gestión del perfil
03-Hace click en “eliminar usuario”	
	04-Muestra mensaje de confirmación
05-Pulsar “aceprar”	
	06-Elimina usuario y toda la información relacionada con el mismo
	07-Redirecciona a la página de login
Poscondiciones:	
Flujo Alternativo	
Usuario	Sistema
05-Pulsar “cancelar”	
	06-Volver al paso 02 del flujo principal

Tabla 19. Caso de uso extendido: Eliminar perfil

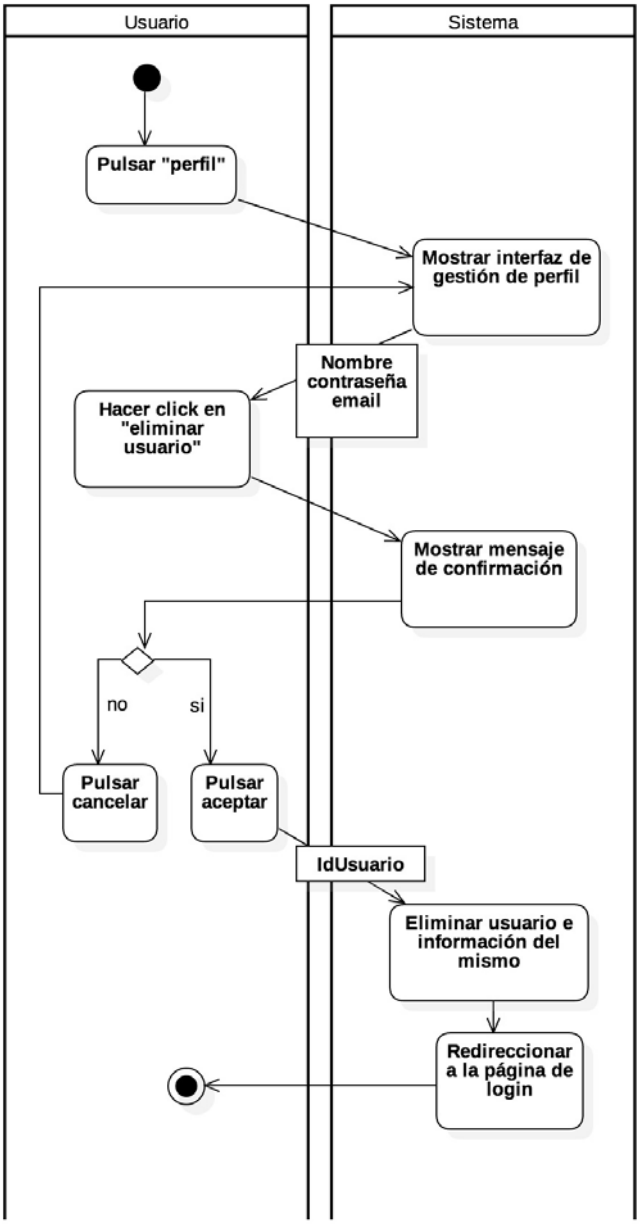


Figura 38. Diagrama de actividad: Eliminar perfil

4.2.13. Caso de uso extendido y diagrama de actividad: Eliminar libro

Caso de Uso	Eliminar libro
Descripción:	Usuario elimina un libro de la biblioteca
Actores:	Usuario
Precondiciones:	El usuario debe haber hecho login y seleccionado una biblioteca. Se visualiza la interfaz miBiblioteca
Flujo Normal	
Usuario	Sistema
01-Pulsar “buscar” del menú de la interfaz miBiblioteca	
	02-Muestra formulario de búsqueda
03-Realizar búsqueda de un libro	
	04-Muestra resultado de la búsqueda
05-Pulsar “eliminar” del libro buscado	
	06-Mensaje de confirmación
07-Confirmar mensaje	
	08-Elimina el libro
	09-Cargar ventana inicio de la interfaz miBiblioteca
Poscondiciones:	
Flujo Alternativo	
Usuario	Sistema
	06-Mensaje de confirmación
07-Cancelar mensaje confirmación	
	08-Volver al paso 03 del flujo principal

Tabla 20. Caso de uso extendido: Eliminar libro

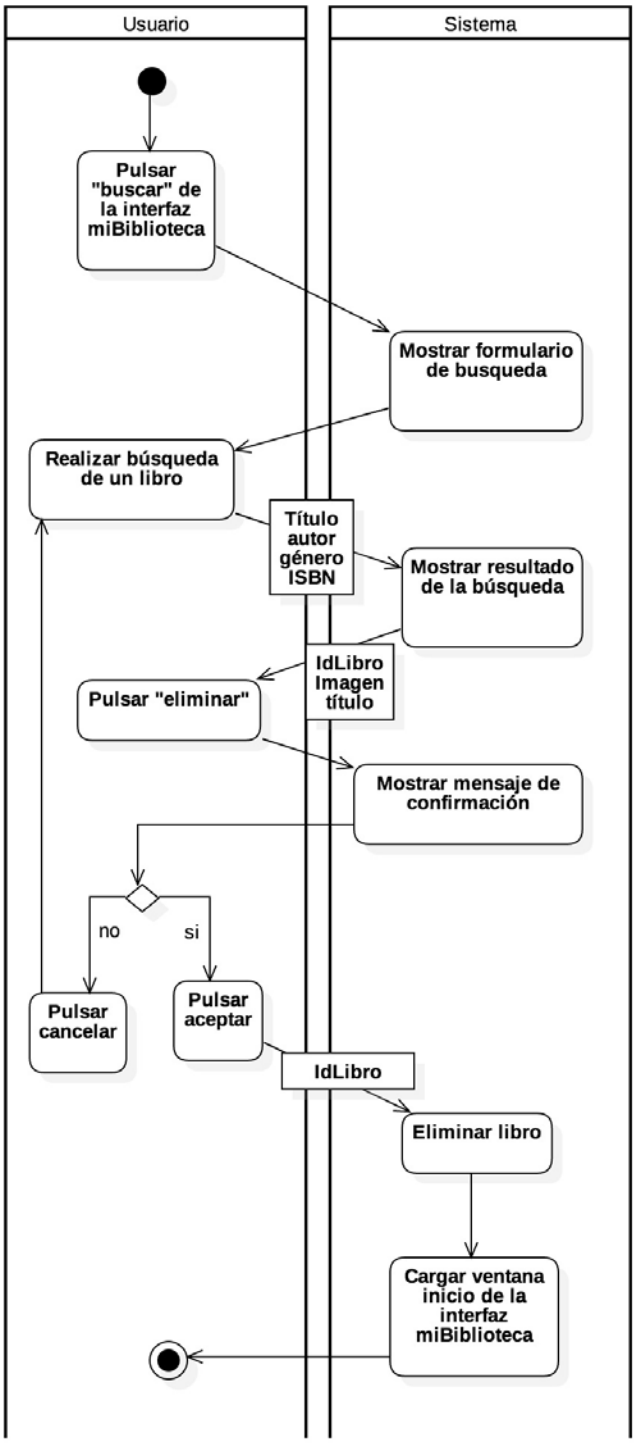


Figura 39. Diagrama de actividad: Eliminar libro

4.2.14. Caso de uso extendido y diagrama de actividad: Visualizar/editar perfil

Caso de Uso	Visualizar/editar perfil
Descripción:	El usuario quiere visualizar su perfil y poder modificarlo
Actores:	Usuario
Precondiciones:	EL usuario tiene que haber hecho login. Se visualiza cualquier ventana de la aplicación
Flujo Normal	
Usuario	Sistema
01- Pulsar “perfil” de la esquina superior derecha	
	02-Mostrar datos del usuario conectado
03-Modificar los datos	
04-Pulsar “editar”	
	05-Mostrar mensaje de confirmación
06-Pulsar aceptar	
	07-Validar los datos
	08-Guardar los nuevos datos y cargar la página de gestión del perfil
Poscondiciones:	
Flujo Alternativo	
Usuario	Sistema
	07-Validar los datos
	08-Mostrar mensaje de error por la validación de las entradas
09-Cerrar mensaje de error	

	10-Volver al paso 03 del flujo principal
Usuario	Sistema
06-Pulsar cancelar	
	07-Volver al paso 03 del flujo principal

Tabla 21. Caso de uso extendido: Visualizar/editar perfil

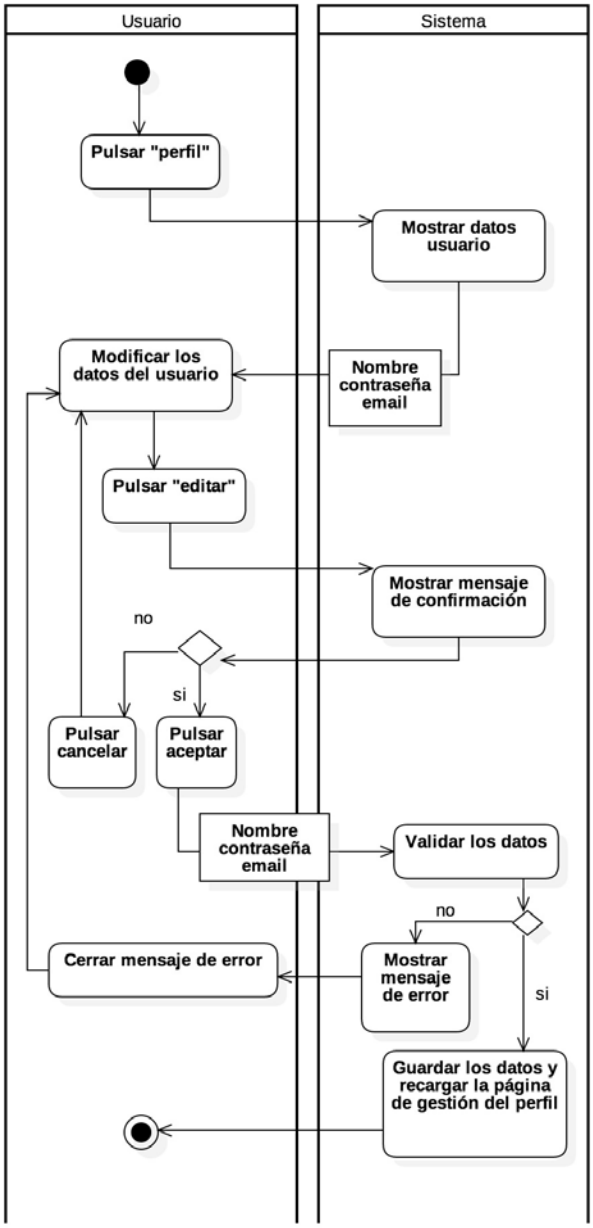


Figura 40. Diagrama de actividad: Visualizar/editar perfil

4.3. Arquitectura del software

4.3.1- Definiciones

4.3.1.1- Arquitectura del software

Paul Clements: La Arquitectura de Software es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se la percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones.

IEEE Std 1471-2000: La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución.

Shaw: La arquitectura software implica la descripción de los elementos desde los que se construye el sistema, interacciones entre esos elementos, patrones que guían su composición y restricciones en estos patrones.

4.3.1.2- Estilo arquitectónico

El estilo arquitectónico es la descripción de los tipos de componentes y un patrón de su control de ejecución y/o transferencia de datos.

4.3.1.3- Modelo de referencia

El modelo de referencia es una división de las funcionalidades junto con los flujos de datos entre las piezas.

4.3.1.4- Arquitectura de referencia

Se define a la arquitectura de referencia como el modelo de referencia planeado para los componentes software (que implementarán las funcionalidades definidas en el modelo de referencia) y los flujos de datos entre componentes.

4.3.2. Estilo arquitectónico seleccionado

Tras realizar un estudio del sistema a implementar, se observa que el flujo de información se realiza mediante peticiones y respuestas, por ello desechamos estilos arquitectónicos como *pipe and filter* debido a que la información no tiene un recorrido único de inicio a fin pasando por diferentes controladores. Por lo tanto creemos que el estilo que más se adecua al sistema a implementar sigue un estilo arquitectónico cliente-servidor, el cual, se puede ver reflejado en la figura 41.

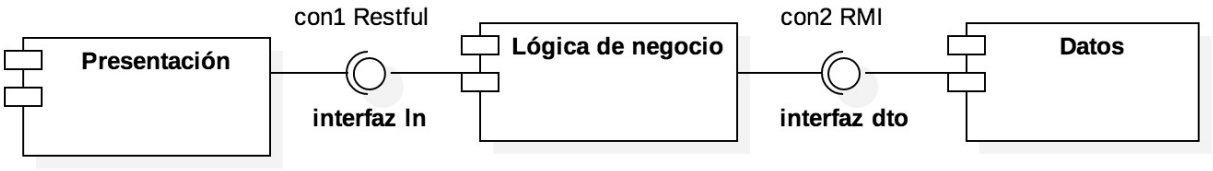


Figura 41. Estilo arquitectónico cliente-servidor seleccionado

4.3.3. Descripción arquitectónica

```
architecture miBiblioteca is {

component_types {

        component Presentacion is extern {miBiblioteca/Presentacion.c2;}
        component LogicaNegocio is extern {miBiblioteca/LogicaNegocio.c2;}
        component Datos is extern {miBilioteca/Datos.c2;}

}

connector_types{
        connector conAPI is {RestFul;}
        connector conDatos is {RMI;}
}


architectural_topology {

        component_instances {
                p:Presentacion;
                l:LogicaNegocio;
                d:Datos;
        }
        connector_instances{
                con1:conAPI;
                con2:conDatos;
        }
        connections{
                connector con1 {p,l;}
```

```
                connector con2 {l,d;}
        }

}

}


component Presentacion is{

"interfaz ln"

        req List <Libros> todosLibros(int idUsuario, int idBiblioteca)
        req List <Libros> buscarLibro (int idUsuario, int idBiblioteca, string titulo)
        req Libros crearLibro (int idUsuario, int idBiblioteca, String titulo, String Autor, String Genero,
      String Saga, String editorial, String Páginas, String ISBN, String fila, String columna, String resumen,
      File imagen, String codigo)
        req Libros buscarPorID(int idUsuario, int idBiblioteca, int idLibro)
        req String editarLibro (int idUsuario, int idBiblioteca, String titulo, String Autor, String Genero,
      String Saga, String editorial, String Páginas, String ISBN, String fila, String columna, String resumen,
      File imagen, String codigo)
        req String eliminarLibro(int idUsuario, int idBiblioteca, int idLibro)
        req String editarPerfil(int idUsuario, String contraseña, String email)
        req Perfil infoPerfil(int idUsuario)
        req String crearUsuario(String nombre, String contraseña, String email)
        req String eliminarUsuario(int idUsuario)
        req Usuario accesoUsuario(String nombre, String contraseña)
        req List <Bibliotecas> todasBibliotecas(int idUsuario)
        req Biblioteca infoBiblioteca (int idUsuario, int idBiblioteca)
        req String eliminarBiblioteca(int idUsuario, int idBiblioteca)
        req String crearBiblioteca(idUsuario, String nombre, String descripcion)
        req String editarBiblioteca(idUsuario, String nombre, String descripcion)

}


component LogicaNegocio is{

"interfaz dto”

        req Connection GetConnection()
        req void close(Connection con)
        req executeQuery(query)

“interfaz ln”

        prov List <Libros> todosLibros(int idUsuario, int idBiblioteca)
        prov List <Libros> buscarLibro (int idUsuario, int idBiblioteca, string titulo)
        prov Libros crearLibro (int idUsuario, int idBiblioteca, String titulo, String Autor, String  Genero,
      String Saga, String editorial, String Páginas, String ISBN, String fila, String columna, String resumen,
      File imagen, String codigo)
```

```
    prov Libros buscarPorID(int idUsuario, int idBiblioteca, int idLibro)
    prov String editarLibro (int idUsuario, int idBiblioteca, String titulo, String Autor, String Genero,
  String Saga, String editorial, String Páginas, String ISBN, String fila, String columna, String resumen,
  File imagen, String codigo)
    prov String eliminarLibro(int idUsuario, int idBiblioteca, int idLibro)
    prov String editarPerfil(int idUsuario, String contraseña, String email)
    prov Perfil infoPerfil(int idUsuario)
    prov String crearUsuario(String nombre, String contraseña, String email)
    prov String eliminarUsuario(int idUsuario)
    prov Usuario accesoUsuario(String nombre, String contraseña)
    prov List <Bibliotecas> todasBibliotecas(int idUsuario)
    prov Biblioteca infoBiblioteca (int idUsuario, int idBiblioteca)
    prov String eliminarBiblioteca(int idUsuario, int idBiblioteca)
    prov String crearBiblioteca(idUsuario, String nombre, String descripcion)
    prov String editarBiblioteca(idUsuario, String nombre, String descripcion)

}

component Datos is {

"interfaz dto"

    prov Connection getConnection(servidor,usuarioDB,passwordDB)
    prov void close(Connection connection)
    prov resultSet prepareStatement(query).executeQuery()


}
```

4.4. Diagrama de clases

En la figura 42 se representa el diagrama de clases diseñado para el servicio.

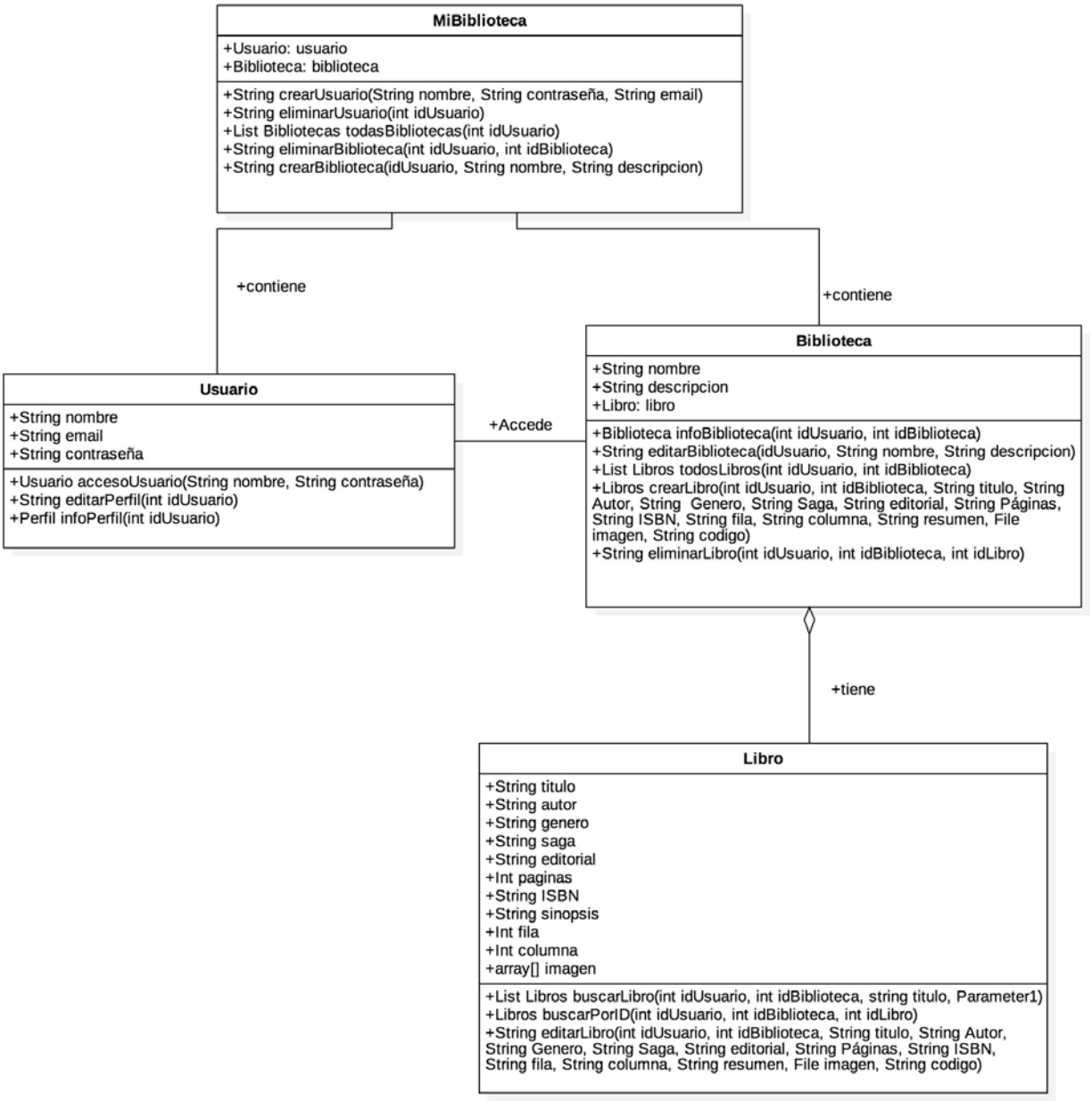


Figura 42. Diagrama de clases

4.5. Diagrama de despliegue

Como se ha visto, se va a realizar un servicio Web basado en REST, encargado de realizar una serie de funciones para la gestión de diferentes bibliotecas.
Cada uno de los métodos implementados en el servicio es “atacado” a través de un número de URIs definidas con anterioridad identificando directamente los recursos creados.

El servicio REST al estar desarrollado en Java, correrá bajo el contenedor de *servlets* Tomcat.

El servicio Web interactúa con dos bases de datos, que estarán replicadas (master-master) para mejorar el rendimiento y la fiabilidad del sistema. Cada base de datos estará alojada en dos máquinas diferentes. Estas bases de datos estarán balanceadas mediante el balanceador de carga HAproxy.

Con el objetivo de hacer uso del servicio Web y probar su correcto funcionamiento, en este caso se ha desarrollado una aplicación web en PHP, HTML y JavaScript, la cual será ejecutado en una nueva máquina bajo un servidor apache.

A nivel aclarativo, las partes que intervienen en el sistema implementado, son las siguientes:

- Aplicación Web
- API REST
- Balanceador
- BBDD1
- BBDD2

Tal como está especificado en la figura 43, el cliente (*Web browser*) consume la aplicación Web, la cual mediante llamadas REST al servicio Web, obtiene la información necesaria en cada momento. A su vez, la API REST para obtener la información pedida, accede a dos bases de datos replicadas y balanceada por HAproxy.

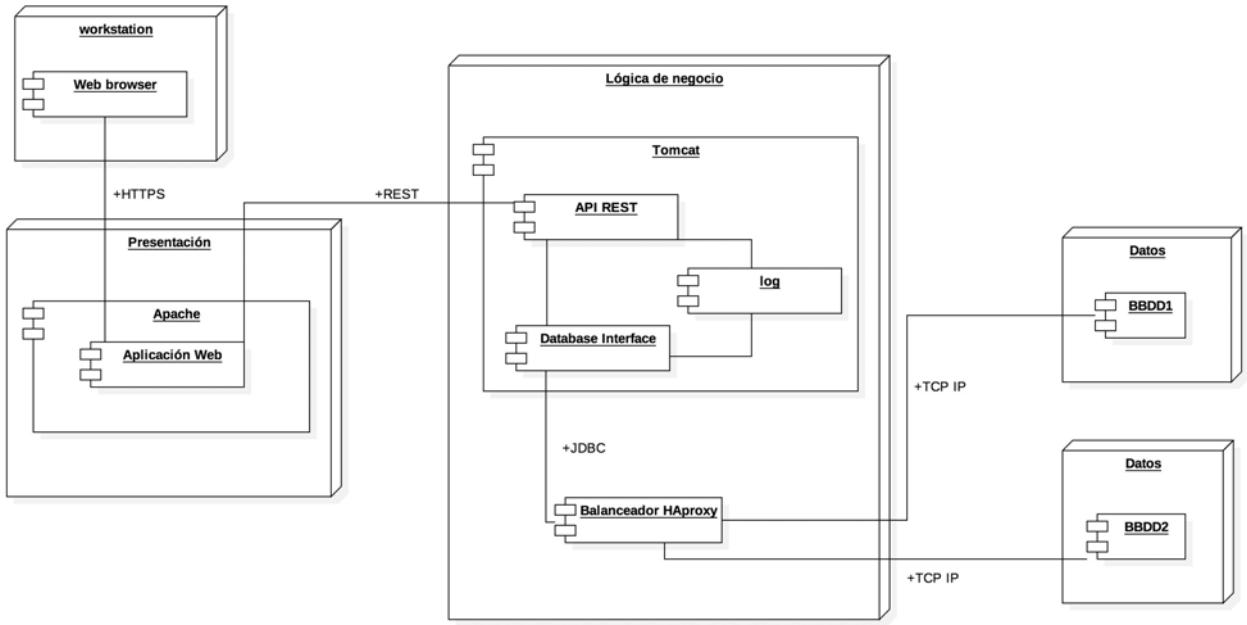


Figura 43. Diagrama de despliegue

5. IMPLEMENTACION Y ALTA DEL SERVICIO

5.1 Configuración básica de un servicio REST

Un servicio basado en Restful se puede desarrollar en diferentes plataformas y lenguajes de programación pero en este apartado nos centraremos en comentar las herramientas utilizadas en este proyecto.

El servicio Web se ha realizado en el lenguaje de programación Java. Para ello, Java ofrece una serie de posibilidades a través del Java Specification Request (JSR) 311. Esta especificación se denomina JAX-RS (*The Java API for RESTful Web Services*) y haremos uso de ella a través del *framework* Jersey, haciendo la programación de estos servicios bastante más sencilla ofreciendo una serie de anotaciones, las más importantes se definen en la tabla 22, para definir la relación de REST con las clases Java desarrolladas.

Otra herramienta que se utilizará es Maven. Como en cualquier lenguaje de programación a veces se necesita utilizar una serie de librerías las cuales, pueden depender de otras. Estas dependencias pueden llegar a ser algo engorrosas por los que gracias a este conjunto de herramientas que Maven ofrece, podemos simplificar esta tarea. Basta con generar un fichero denominado pom.xml y agrupar todas las dependencias que necesitemos en él. De esta manera, estarán siempre accesibles de una manera más sencilla en nuestro proyecto.

Anotaciones	Descripción
@PATH()	Ruta de acceso relativa
@POST, @GET, @PUT,@DELETE	Especifican el tipo de petición HTTP de un recurso
@HeaderParam	Enlaza el parámetro a un valor de cabecera HTTP
@FormParam	Enlaza el parámetro a un valor de formulario
@FormDataParam	Enlaza un parámetro con un fichero (imágenes)
@Produces(MediaType.TEXT_PLAIN[, more-types])	Especifican los tipos de medios MIME de respuesta
@Consumes(type[, more-types])	Especifica los tipos de medios de petición aceptados
@PathParam	Enlaza el parámetro a un segmento de ruta

Tabla 22. Anotaciones JAX-RS

La plataforma utilizada para el desarrollo del servicio es eclipse. A continuación se visualizarán los pasos a seguir para configurar desde cero un servicio básico basado en REST en eclipse utilizando la API JAX-RS y la herramienta Maven y así saber de manera simplificada como se ha realizado el desarrollo de nuestro servicio.

Los pasos a seguir para configurar un proyecto Restful se detallan a continuación:

Paso 1.

Creamos un nuevo proyecto dinámico como muestra la figura 44.

File => New => Dynamic Web Project. nombramos el proyecto como “PrimerservicioREST”.

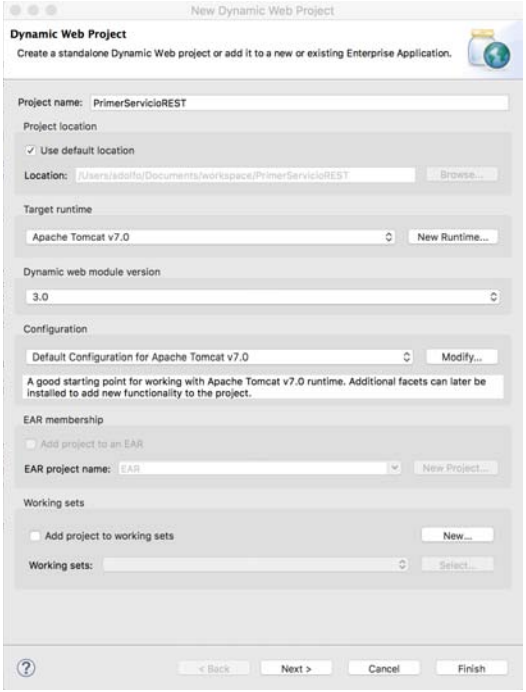


Figura 44. Nuevo proyecto dinámico (eclipse)

Paso 2.

Generamos el fichero web.xml, el componente de despliegue de aplicaciones Java, donde se detalla la información necesaria para desplegar una aplicación Web. Se encuentra bajo el directorio WEB-INF.

Botón derecho sobre proyecto => Java EE Tools > Generate deployment descriptor Stub

Paso 3.

Convertimos el proyecto en un proyecto específico de maven y de esta manera generamos un nuevo fichero pom.xml, comentado con anterioridad.

Botón derecho sobre proyecto => Cofigure => Convert to Maven Proyect

Paso 4.

Añadimos las dependencias necesarias al fichero pom.xml. A continuación se muestra un ejemplo de un fichero pom.xml básico.

```xml
<?xml version="1.0"?>
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd" xmlns="http://
maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <modelVersion>4.0.0</modelVersion>
  <groupId>gestionBibliotecas</groupId>
  <artifactId>gestionBibliotecas</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>
  <dependencies>
    <dependency>
      <groupId>asm</groupId>
      <artifactId>asm</artifactId>
      <version>3.3.1</version>
      <scope>compile</scope>
    </dependency>
    <dependency>
      <groupId>com.sun.jersey</groupId>
      <artifactId>jersey-bundle</artifactId>
      <version>1.19</version>
      <scope>compile</scope>
    </dependency>
  </dependencies>
</project>
```

Paso 5.

Modificar el fichero web.xml según nuestras necesidades, en el se define el *servelet container* e indicamos la URI base de nuestro servicio (<url-pattern>/miServicio/*</url-pattern>).

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://
java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
          xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://
java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
          version="3.0">
          <display-name>PrimerServicioREST</display-name>
          <servlet>
    <servlet-name>Jersey Web Application</servlet-name>
    <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</
servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Jersey Web Application</servlet-name>
    <url-pattern>/miServicio/*</url-pattern>
  </servlet-mapping>
</web-app>
```

Paso 6.

Generamos una clase para el servicio a implementar.

```
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("/bienvenida")
public class primerServicio {

        @GET
        @Produces(MediaType.TEXT_PLAIN)
        public String Hola(){

                        return "Mensaje de bienvenida al maravilloso mundo de REST";

        }
}
```

Paso 7.

Desplegamos la aplicación mediante el servidor Tomcat

Botón derecho sobre proyecto => run us => run on server

Paso 8.

Probar el funcionamiento del servicio.

Una vez que la aplicación ya se está ejecutando en el servidor, siguiendo las rutas establecidas anteriormente podemos consumir el servicio a través de la siguiente ruta:

http://localhost:8080/PrimerServicioREST/miServicio/bienvenida

Mediante esta URL podemos acceder a la primera clase creada y relacionada mediante el path "/bienvenida" y método GET.

El resultado de la operación se puede ver en la figura 45.

Figura 45. Resultado primer servicio REST implementado

5.2 Securización del servicio: JWT

La securización de los servicios REST empieza a ser un tema muy importante y cada vez con más frecuencia se pide autenticación para poder acceder a cualquier dato que ofrece el servicio. Con ataques cada vez más frecuentes, la seguridad de estos servicios se basa en prevenir alteraciones en los mensajes transmitidos, *sniffers* de red que permitan leer los mensajes, etc. Pero también se cubren otros tipos de seguridad como pueden ser dar permisos por usuario a determinadas funciones del servicio.

Hoy en día existen diferentes soluciones para cubrir los aspectos más importantes: autenticación, autorización y encriptación. Existen diferentes métodos para autenticarse ante un servicio REST y cubrir estos puntos críticos.

Autenticación

Nos sirve para validar la identidad de una persona. Los diferentes protocolos existentes son los siguientes:

- Autenticación básica —> Consiste en enviar el usuario y contraseña codificados en Base64 sobre HTTP. Es el protocolo más simple

- Digest authentication —> No se usa mucho pero es un pequeño avance del anterior para resolver el problema de enviar el usuario y contraseña en claro. En este caso se envía el usuario y contraseña con función hash MD5

- Autenticación mediante certificado X.509 —> La forma más segura de realizar una autenticación sería mediante el envío de un certificado personal sobre HTTPS

Autorización

Con la autenticación se consigue verificar la identidad de un usuario. Con la autorización se controlan los permisos que un usuario tiene para realizar una acción.

Existen muchos métodos para poder realizar este tipo de control pero el servicio realizado en este proyecto esta desarrollado en Java y utilizando una API determinada, JAX-RS, por lo que nos centraremos en este método.

Utilizando Java, la forma de poder realizar una autorización es modificando el fichero comentado con anterioridad, web.xml y añadir para cada recurso que se quiera securizar mediante autorización un rol determinado, que se deberá enviar en cada petición.

```
<security-role>
  <role-name>admin</role-name>
</security-role>
```

Encriptación

Para que los datos no puedan ser leídos, en caso de ser interceptados se recomienda el uso del protocolo HTTPS, utilizando SSL para encriptar los mensajes que se transmiten.

JWT (JSON Web Tokens)

Normalmente, en la mayoría de las aplicaciones cliente-servidor, era común que el servidor guardara una sesión de un cliente cuando este se autenticaba y de esta manera poder reconocer quién es la persona que se conecta instantáneamente. Esto consume más recursos, es difícil de escalar y no se debe olvidar que una de las principales características de REST es la de no tener estado.

Una de las últimas tendencias en el desarrollo Web es la autenticación mediante *tokens*, con la que se puede cubrir todo lo que se ha comentado anteriormente, autenticación, autorización, encriptación (para la que se recomienda el uso de SSL) y el fundamento básico de REST, no mantener estado.

El funcionamiento de una autenticación mediante *tokens* es el siguiente:

1- El cliente se autentica en la aplicación mediante usuario y contraseña
2- El servicio genera un *token* determinado con diferente información del usuario y lo envía
3- El cliente envía y almacena el *token* generado por el servicio
4- El cliente en cada petición que realice envía el *token* que posee
5- El servicio identifica al usuario a través del *token* recibido y permite o deniega el acceso

El estándar en Java para este tipo de autenticación es el JSON Web Token (JWT):

Un *token* JWT está compuesto por tres *strings* separados por un punto (.):

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI1NGE4Y2U2MThlOTFiMGIxMzY2NWUyZjkiLCJpYXQiOiIxNDI0MTgwNDg0IiwiZXhwIjoiMTQyNTM5MDE0MiJ9.yk4nouUteW54F1HbWtgg1wJxeDjqDA_8AhUPyjE5K0U

Las tres partes de las que está compuesto el *token* son las siguientes:

- **HEADER**: El primer *string* se corresponde con la cabecera y contiene los metadatos del *token* formado por el tipo y codificación utilizada. La codificación utilizada suele ser HMAC SHA256.

```
{
  “typ”: “JWT”,
  "alg": "SHA256"
}
```

Como se visualiza en el ejemplo anterior, la cabecera indica el tipo de *token*, JWT y la codificación utilizada, SHA256.

- **PAYLOAD:** El *payload* lo conforman un conjunto de datos denominados *JWT claims*. Se especifican diferentes atributos que identificaran al *token*.

  Claims reservados: Estos *claims* se encuentran registrados en el *IANA JSON Web Token Claims registry*. Estos datos no son obligatorios pero ayudan a identificar un conjunto de datos básicos. Existen varios valores. Algunos de los más importantes son los siguientes:

  - iss —> responsable de la creación del *token*
  - sub —> asunto del *token*
  - aud —> audiencia a la que va dirigido
  - exp —> tiempo de expiración

  Claims públicos: Estos datos se describen con nombres resistentes a colisiones incluyendo una URI junto al nombre.

  Claims privados: Estos datos pueden ser definidos por el propio desarrollador y se utilizan para definir IDs, roles de usuario o cualquier otra información que se requiera.

A continuación se visualiza cómo se define el *payload* de un *token*:

```
{
  "iss": "avaldes.com",
  "exp": 1450626101,
  "jti": "qicrd_B9So9MGEKXZV99Fg",
  "iat": 1450625501,
  "nbf": 1450625381,
  "sub": "apacheuser",
  "roles": [
    "client",
    "admin"
  ]
}
```

- **FIRMA**: la última parte del *token* consiste en una codificación en BASE64 tanto del HEADER como del PAYLOAD que junto a una clave privada, le realizamos un *hash* utilizando cualquier código de autenticación de mensajes (MAC) como puede ser HMAC RSA SHA-256 para encriptar y mantener la integridad del mensaje.

```
HMAC-RSA-SHA-256(
  base64encode(header) + "." +
  base64encode(payload),
  Private Key
)
```

La implementación del token de este proyecto se puede visualizar en el anexo b3.

5.3. Diseño y planteamiento del servicio implementado: miBiblioteca

Como hemos visto, todo servicio REST debe seguir unos principios arquitectónicos. Los pasos realizados para que nuestra aplicación cumpla con todos los requisitos son los siguientes:

1. Analizar el modelo de datos del servicio

   El modelo de datos, que se puede visualizar en la sección de arquitectura del software, nos sirve para identificar los recursos principales del servicio.
   En este caso, podemos identificar tres claros recursos: usuario, biblioteca y libro.

2. Definir un conjunto de URIs que representen los puntos de entrada en nuestro sistema cumpliendo así con el principio de direccionamiento.

   Las URIs para poder acceder al servicio se listan a continuación:

   - /gestionBiblioteca/bibliotecas/miBiblioteca/login
   - /gestionBiblioteca/bibliotecas/miBiblioteca/{idBiblioteca}/libros
   - /gestionBiblioteca/bibliotecas/miBiblioteca/{idBiblioteca}/libros/{idLibro}
   - /gestionBiblioteca/bibliotecas/miBiblioteca/{idBiblioteca}/libros/nombre/{nombreLibro}
   - /gestionBiblioteca/bibliotecas/biblioteca
   - /gestionBiblioteca/bibliotecas/{idBiblioteca}
   - /gestionBiblioteca/bibliotecas/usuario
   - /gestionBiblioteca/bibliotecas/usuario/{nombre}
   - /gestionBiblioteca/bibliotecas/usuario/{idUsauario}

3. Definir el formato de los datos que se van a intercambiar cumpliendo el principio de REST: orientado a la representación.

   Independientemente de que con REST se pueda enviar diferentes tipos de datos, usualmente las representaciones de recursos son texto y se debe elegir un formato de representación para estos casos. XML y JSON son los más usuales.
   Por ser algo más ligero y versátil y pudiéndolo utilizar en aplicaciones programadas en JavaScript, el formato elegido para enviar las representaciones de los recursos del servicio es JSON.

4. Definir los métodos HTTP que estarán permitidos para cada una de las URIs declaradas anteriormente y qué es lo que van a hacer

   - **POST /gestionBiblioteca/bibliotecas/miBiblioteca/login** —> Registro en la aplicación
   - **GET /gestionBiblioteca/bibliotecas/miBiblioteca/{idBiblioteca}/libros** —> Obtener todos los libros de una biblioteca determinada
   - **POST /gestionBiblioteca/bibliotecas/miBiblioteca/{idBiblioteca}/libros** —> Insertar un nuevo libro en una biblioteca determinada
   - **GET /gestionBiblioteca/bibliotecas/miBiblioteca/{idBiblioteca}/libros/{idLibro}** —> Obtener la información de un libro de una biblioteca mediante si identificador
   - **POST /gestionBiblioteca/bibliotecas/miBiblioteca/{idBiblioteca}/libros/{idLibro}** —> Editar la información de un libro ya creado indicando su identificación
   - **DELETE /gestionBiblioteca/bibliotecas/miBiblioteca/{idBiblioteca}/libros/{idLibro}** —> Eliminar un libro determinado a través de su identificador

- **GET /gestionBiblioteca/bibliotecas/miBiblioteca/{idBiblioteca}/libros/nombre/{nombreLibro}** —> Búsqueda de un libro mediante su nombre
- **GET /gestionBiblioteca/bibliotecas/biblioteca** —> Obtener todas las bibliotecas de un usuario. La información de un usuario se coge mediante el token recibido
- **POST /gestionBiblioteca/bibliotecas/biblioteca** —> Crear una nueva biblioteca
- **GET /gestionBiblioteca/bibliotecas/{idBiblioteca}** —> Obtener la información de una biblioteca creada mediante si identificación
- **POST /gestionBiblioteca/bibliotecas/{idBiblioteca}** —> Editar la información de una biblioteca
- **DELETE /gestionBiblioteca/bibliotecas/{idBiblioteca}** —> Eliminar una biblioteca del sistema
- **GET /gestionBiblioteca/bibliotecas/usuario** —> Obtener una lista con todos los usuarios del sistema
- **POST /gestionBiblioteca/bibliotecas/usuario** —> Crear una nueva cuenta en el sistema
- **GET /gestionBiblioteca/bibliotecas/usuario/{nombre}**—>Obtener la información del usuario registrado
- **POST /gestionBiblioteca/bibliotecas/usuario/{nombre}** —> Editar la información relacionada con el usuario registrado
- **DELETE /gestionBiblioteca/bibliotecas/usuario/{idUsauario}** —> Eliminar al usuario registrado del sistema

Una vez definida nuestra API y comprobar que casa con todas las características que definen el estilo REST, se puede proceder a su implementación.

A modo de ejemplo, en el anexo b1, se muestra un ejemplo de la implementación de uno de los métodos del servicio.

5.4. Implementación del cliente Web

Para implementar el cliente Web con el fin de probar el servicio desarrollado, se ha utilizado las siguientes tecnologías:

- HTML y bootstrap

  HTML (HyperText Markup Language) es el lenguaje para la generación de páginas Web mas conocido. El cliente se ha implementado con el framework de código abierto bootstrap, a la hora de realizar el diseño de la aplicación Web.

- JavaScript

  Es un lenguaje de programación interpretado orientado a objetos. Se utiliza principalmente en el lado del cliente, implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas. Se ha utilizado en el cliente para mejor la experiencia del usuario con mensajes informativos y despliegue de menús.

- AJAX

  AJAX (**A**synchronous **J**avaScript **A**nd **X**ML), técnica de desarrollo Web que permite la comunicación con servicios Web intercambiando ficheros XML y JSON, entre otros. Se encarga de interpretar las respuestas que el servicio envía. Se ha utilizado en alguna llamada al servicio implementado.

- PHP

  Lenguaje de programación del lado del servidor utilizado en el desarrollo Web. El cliente se ha programado con este lenguaje para la mayoría de las interacciones realizadas con el servicio.

- cURL

  Software con una biblioteca y un interprete de comandos utilizado para la transferencia de archivos. La mayoría de lenguajes de programación tienen librerías para poder utilizar este software. En este caso cURL se ha programado en PHP para la mayoría de las transferencias de datos entre el cliente y el servidor.
  A modo de ejemplo, en el anexo b2 se muestran dos ejemplos de la utilización de cURL con php

5.5. Balanceo de carga: HAproxy y base de datos MySQL con replicación

A continuación veremos someramente cómo configurar dos bases de datos para permitir una réplica master-master y la configuración básica del balanceador de carga HAProxy.

La réplica master-master permite que los datos que se añadan a cualquiera de las bases de datos creadas, se añadirá a las siguientes, manteniendo de esta manera la misma información en todas las bases de datos. El objetivo de este tipo de réplica es conseguir redundancia y eficiencia cuando creamos una conexión con una base de datos.

Vamos a utilizar dos servidores; Server A: 192.168.1.40 y Server B: 192.168.1.41

Los pasos a seguir son los siguientes: instalar y configurar MySQL tanto en el servidor A como en el B, posteriormente completar la réplica de datos en el servidor A.

1-Instalación y configuración de MySQL en el servidor A.

```
sudo apt-get install mysql-server mysql-client
```

Por defecto MySQL solo acepta conexiones a *localhost*, por lo que se debe configurar el siguiente archivo /etc/mysql/my.cnf, como se ve a continuación:

```
server-id          = 1
log_bin            = /var/log/mysql/mysql-bin.log
binlog_do_db       = miBiblioteca
# bind-address     = 127.0.0.1
```

Una vez realizado los cambios requeridos, reiniciamos el servicio MySQL y accedemos a la base de datos (mysql -u root -p) para crear un usuario y darle permisos para realizar la réplica:

```
create user 'replicator'@'%' identified by
'password';

grant replication slave on *.* to
'replicator'@'%';
```

Tras realizar estos pasos ejecutaremos el siguiente comando, **show master status;**
Este comando nos va a facilitar información sobre los archivos de *logs* binarios del master (servidor A), como podemos ver en la figura 46, que más adelante necesitaremos indicar en el servidor B.

```
mysql> show master status;
+------------------+----------+--------------+------------------+-------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set |
+------------------+----------+--------------+------------------+-------------------+
| mysql-bin.000008 |     1057 | miBiblioteca |                  |                   |
+------------------+----------+--------------+------------------+-------------------+
1 row in set (0.00 sec)
```

Figura 46. Master status servidor 1

2-Instalación y configuración de MySQL en el servidor B.

El proceso de instalación del servicio MySQL es exactamente el mismo que para el servidor A, en este caso el archivo /etc/mysql/my.cnf quedaría de la siguiente manera:

```
server-id          = 2
log_bin            = /var/log/mysql/mysql-bin.log
binlog_do_db       = miBiblioteca
# bind-address     = 127.0.0.1
```

A continuación crearemos también un usuario con derecho para realizar una réplica y crearemos la base de datos a utilizar:

```
create user 'replicator'@'%' identified by 'password';
create database example;
grant replication slave on *.* to 'replicator'@'%';
```

Una vez que está todo configurado, tenemos que indicar a cada servidor cuál es la máquina donde se tiene que replicar. Para ello, necesitamos la información del estado del servidor A, conseguida con anterioridad y realizar lo siguiente:

```
slave stop;
CHANGE MASTER TO MASTER_HOST = '192.168.1.40', MASTER_USER
= 'replicator', MASTER_PASSWORD = 'password', MASTER_LOG_FILE
= 'mysql-bin.000008', MASTER_LOG_POS = 1057;
slave start;
```

Por último en el servidor D, veremos el estado del master D (figura 47), cuya información necesitaremos para completar la configuración en el servidor D

```
mysql> show master status;
+------------------+----------+--------------+------------------+-------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set |
+------------------+----------+--------------+------------------+-------------------+
| mysql-bin.000009 |      154 | miBiblioteca |                  |                   |
+------------------+----------+--------------+------------------+-------------------+
1 row in set (0.01 sec)
```

Figura 47. Master status servidor 2

3-Completar réplica en servidor A.

```
slave stop;
CHANGE MASTER TO MASTER_HOST = '192.168.1.41', MASTER_USER =
'replicator', MASTER_PASSWORD = 'password', MASTER_LOG_FILE =
'mysql-bin.000009', MASTER_LOG_POS = 154;
slave start;
```

Una vez hecho esto, las dos bases de datos están configuradas correctamente para soportar una réplica master-master.

Configuración Balanceador HAproxy.

Para poder configurar el balanceador de carga identificaremos las siguientes máquinas:

Hostname: haproxy; IP: 192.168.1.42
Hostname: mysql-1; IP: 192.168.1.40
Hostname: mysql-2; IP: 192.168.1.41

Los pasos a seguir para la correcta configuración son los siguientes: prepararemos lo servidores MySQL configurados con anterioridad, instalaremos un cliente MySQL en el servidor haproxy, instalaremos haproxy y, por último, configuraremos haproxy.

1- Preparación servidores MySQL

Al estar replicados master-master basta con realizar lo siguiente en una sola máquina. Con el fin de hacer una revisión del estado del servidor deberemos crear un usuario haproxy con privilegios de *root*.

```
root@mysql-1# mysql -u root -p -e "INSERT INTO mysql.user (Host,User) values ('192.168.1.42','haproxy_check'); FLUSH PRIVILEGES;

root@mysql-1# mysql -u root -p -e "GRANT ALL PRIVILEGES ON *.* TO 'haproxy_root'@'192.168.1.42' IDENTIFIED BY 'password' WITH GRANT OPTION; FLUSH PRIVILEGES"
```

2- Instalación del cliente MySQL

Se debe de instalar un cliente MySQL en el servidor donde estará alojado el haproxy para probar la conectividad con los servidores de base de datos.

```
root@haproxy# apt-get install mysql-client
```

Podemos comprobar que podemos ejecutar un comando en cualquiera de las bases de datos instaladas con el usuario haproxy de la siguiente manera:

```
root@haproxy# mysql -h 192.168.1.40 -u haproxy_root -p -e "SHOW DATABASES"
```

3- Instalación HAProxy

```
root@haproxy# apt-get install haproxy
root@haproxy# sed -i "s/ENABLED=0/ENABLED=1/" /etc/default/haproxy
```

4- Configuración del servidor HAProxy

Una vez instalado el balanceador, procedemos a modificar su fichero de configuración, /etc/haproxy/haproxy.cfg, de la siguiente manera:

```
global
    log 127.0.0.1 local0 notice
    user haproxy
    group haproxy

defaults


    log global
    retries 2
    timeout connect 3000
    timeout server 5000
    timeout client 5000

listen mysql-cluster
    bind 127.0.0.1:3306
    mode tcp
    option mysql-check user haproxy_check
    balance roundrobin
    server mysql-1 192.168.1.40:3306 check
    server mysql-2 192.168.1.41:3306 check

listen 0.0.0.0:8080
    mode http
    stats enable
    stats uri /
    stats realm Strictly\ Private
    stats auth root:root
    stats auth user:password
```

Para finalizar solo tendremos que iniciar el servicio, **service haproxy start**. De esta manera ya estaría configurado el balanceador con dos bases de datos con réplica master-master.
Esta configuración está realizada para trabajar con balanceamiento de carga *roundrobin* con pesos iguales para las dos bases de datos.

El código de las tablas generadas para la base de datos del proyecto, se pueden visualizar en el anexo b4.

6-VERIFICACIÓN Y VALIDACIÓN DEL SERVICIO. PRUEBAS

6.1. Verificación del servicio REST

6.1.1. Login y generación de token de sesión

El servicio recibe en formato de texto un usuario y una contraseña con el fin de comprobar su existencia y permitir el acceso a la aplicación.
El servicio tras recibir un usuario y contraseña correctos, devuelve un código HTTP 200 (OK) de confirmación y en la cabecera *Authorization* envía el *Token* de sesión para ese usuario como podemos visualizar en la figura 48.
En la figura 49 se visualiza el caso contrario, usuario y contraseña no son correctos, donde el servidor devuelve un código HTTP 401 (no autorizado) y el por qué.

```
POST http://localhost:8080/gestionBibliotecas/bibliotecas/miBiblioteca/login?usuario=elenita&contrasenia=elenita
Content-Type: application/x-www-form-urlencoded

 -- response --
201 Creado
Server:  Apache-Coyote/1.1
Authorization:
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwianRpIjoiNmMyNDRlNTItODNlOS00MDY5LWFhMDMtNDIyYWE4Yjk4N2JiIiwiZX
hwIjoxNDg0MDQxNDQ5LCJub21icmUiOiJlbGVuaXRhIiwiY29udHJhc2XDsWEiOiJlbGVuaXRhIiwiaWQiOjgyfQ.D4bkXmFigNloe8b9OZblp
n4uoPlIlnBq5UouM6Fi2mI
Content-Type:  application/json
Transfer-Encoding:  chunked
Date:  Tue, 10 Jan 2017 09:34:09 GMT

Usuario autenticado correctamente
```

Figura 48. Prueba login y token 1

```
POST http://localhost:8080/gestionBibliotecas/bibliotecas/miBiblioteca/login?usuario=elenitaa&contrasenia=elenita
Content-Type: application/x-www-form-urlencoded

 -- response --
401 No Autorizado
Server:  Apache-Coyote/1.1
Content-Type:  application/json
Transfer-Encoding:  chunked
Date:  Tue, 10 Jan 2017 09:36:43 GMT

Usuario o contraseña no son validos o no tiene permisos
```

Figura 49. Prueba login y token 2

6.1.2. Crear un usuario

Se envía al servicio los datos requeridos de registro: usuario, contraseña e email. Al introducir unos datos válidos y realizarse el registro correctamente, el servicio devuelve un mensaje HTTP con el código 201 (creado) al igual que un nuevo *token* generado como podemos observar en la figura 50.


```
POST http://localhost:8080/gestionBibliotecas/bibliotecas/usuario?usuario=adolf&contrasenia=adolf&email=adolf%40gmail.com
Authorization:
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwianRpIjoiNmMyNDRlNTItODNlOS00MDY5LWFhMDMtNDIyYWE4Yjk4N2JiIiwiZX
hwIjoxNDg0MDY3MzU2LCJub21icmUiOiJhZG9sZm8iLCJjb250cmFzZcOxYSI6ImFkb2xmbyIsImlkIjo4NH0.Omk2L5rRhhG-
D-eysYfHdBmDMnfjD72KaTeY66rjyYI
Content-Type: application/x-www-form-urlencoded

 -- response --
201 Creado
Server:  Apache-Coyote/1.1
Authorization:
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwianRpIjoiNmMyNDRlNTItODNlOS00MDY5LWFhMDMtNDIyYWE4Yjk4N2JiIiwiZX
hwIjoxNDg0MDY3NDMxLCJub21icmUiOiJhZG9sZiIsImNvbnRyYXNlw7FhIjoiYWRvbGYiLCJpZCI6ODV9.Muu6WiPCM8DK-04qzneKo-
JjySbGfmlPfBz0BaqhhXE
Content-Type:  application/xml
Transfer-Encoding:  chunked
Date:  Tue, 10 Jan 2017 16:47:11 GMT

Usuario autenticado correctamente
```


Figura 50. Prueba creación de usuario

6.1.3. Eliminar un usuario

Para la eliminación de un usuario, como se especificó anteriormente se le pasa al servicio el identificador del mismo en la URL. Realizando esta operación de manera correcta, podemos observar en la figura 51 cómo el servicio envía un código de respuesta 200 y como mensaje de texto, el resultado de la operación. En caso de producirse un error se ha comprobado que se devuelve un código de error 500 por existir un problema en la base de datos.
A su vez se ha confirmado en la base de datos que tras eliminar un usuario (el 74 en este caso) se eliminan todas las bibliotecas y los libros de las mismas como se comprueba en la imagen 52.


```
DELETE http://localhost:8080/gestionBibliotecas/bibliotecas/usuario/74
Authorization:
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwianRpIjoiNmMyNDRlNTItODNlOS00MDY5LWFhMDMtNDIyYWE4Yjk4N2JiIiwiZX
hwIjoxNDg0MDY3MzU2LCJub21icmUiOiJhZG9sZm8iLCJjb250cmFzZcOxYSI6ImFkb2xmbyIsImlkIjo4NH0.Omk2L5rRhhG-
D-eysYfHdBmDMnfjD72KaTeY66rjyYI

 -- response --
200 OK
Server:  Apache-Coyote/1.1
Content-Type:  application/json
Transfer-Encoding:  chunked
Date:  Wed, 11 Jan 2017 07:33:11 GMT

Recurso eliminado correctamente
```


Figura 51. prueba eliminar usuario

idUsuario
68
75
81
82
84
85
NULL

idBibliotecas	idUsuario
55	75
57	76
58	76
59	68
60	81
61	81
62	82
63	84

idLibros	idBiblioteca
87	55
88	55
89	55
90	55
91	55
92	55
93	55
94	55
95	62
96	62

Figura 52. Estado base de datos usuario eliminado

6.1.4. Obtener información de un usuario buscado por nombre

Con esta prueba se va a comprobar que el servicio devuelve toda la información que contiene de un usuario. Para ello, al servicio le enviamos el nombre del usuario a buscar si la búsqueda es correcta. Se verifica, como se puede ver en la figura 53, que el servicio envía un código de respuesta 200 junto con el contenido del resultado en formato JSON. En caso de no encontrar un usuario, el resultado que devuelve es NULL.


```
GET http://localhost:8080/gestionBibliotecas/bibliotecas/usuario/adolfo
Authorization:
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwianRpIjoiNmMyNDRlNTItODNlOS00MDY5LWFhMDMtNDIyYWE4Yjk4N2JiIiwiZX
hwIjoxNDg0MDY3MzU2LCJub21icmUiOiJhZG9sZm8iLCJjb250cmFzZcOxYSI6ImFkb2xmbyIsImlkIjo4NH0.Omk2L5rRhhG-
D-eysYfHdBmDMnfjD72KaTeY66rjyYI

 -- response --
200 OK
Server:  Apache-Coyote/1.1
Content-Type:  application/json
Transfer-Encoding:  chunked
Date:  Wed, 11 Jan 2017 08:18:21 GMT

{"contrasenia":"adolfo","email":"adolfo@gmail.com","idUsuario":"84","nombre":"adolfo"}
```


Figura 53. Prueba llamada obtención información de un usuario

6.1.5. Modificar la información de un usuario

El servicio recibe la información modificada de un usuario ya creado en el sistema: nombre, contraseña e email. En la URL para invocar el método requerido se envía el nombre original del usuario para su modificación. Como podemos ver en la figura 54, se realiza una modificación satisfactoria donde el servicio devuelve un código de respuesta 201 y un nuevo *token* generado para mantener la sesión con los cambios realizados. Se verifica a su vez que la información en la base de datos se ha cambiado correctamente.


```
POST http://localhost:8080/gestionBibliotecas/bibliotecas/usuario/adolfo?usuario=adolfoCambios&contrasenia=adolfoCambios&
email=adolfoCambios%40gmail.com
Authorization:
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwianRpIjoiNmMyNDRlNTItODNlOS00MDY5LWFhMDMtNDIyYWE4Yjk4N2JiIiwiZX
hwIjoxNDg0MDY3MzU2LCJub21icmUiOiJhZG9sZm8iLCJjb250cmFzZcOxYSI6ImFkb2xmbyIsImlkIjo4NH0.Omk2L5rRhhG-
D-eysYfHdBmDMnfjD72KaTeY66rjyYI
Content-Type: application/x-www-form-urlencoded

 -- response --
201 Creado
Server:  Apache-Coyote/1.1
Authorization:
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwianRpIjoiNmMyNDRlNTItODNlOS00MDY5LWFhMDMtNDIyYWE4Yjk4N2JiIiwiZX
hwIjoxNDg0MTI1ODY0LCJub21icmUiOiJhZG9sZm9DYW1iaW9zIiwiY29udHJhc2XDsWEiOiJhZG9sZm9DYW1iaW9zIiwiaWQiOjg0fQ.xSd
LwPezYuyfDvHySZbBFjct1rWO1vWdau-VLIox0MA
Content-Type:  application/json
Transfer-Encoding:  chunked
Date:  Wed, 11 Jan 2017 09:01:04 GMT

Informacion editada correctamente
```


Figura 54. Prueba llamada edición del perfil

6.1.6. Buscar todas las bibliotecas de un usuario

El objetivo es verificar que el servicio devuelve una lista con todas las bibliotecas que un usuario del sistema posee. Como podemos ver en la figura 55, si la consulta se ha realizado de manera satisfactoria el servicio envía un código de respuesta 200 junto a un archivo JSON con la información de todas las bibliotecas que posee ese usuario. Además, se reenvía el nuevo *token* generado de manera correcta.
El servicio recibe el *token* de sesión del usuario para poder comprobar la validez y tiempo de conexión así como la información necesaria para la consulta.


```
GET http://localhost:8080/gestionBibliotecas/bibliotecas/biblioteca
Authorization:
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwianRpIjoiNmMyNDRlNTItODNlOS00MDY5LWFhMDMtNDIyYWE4Yjk4N2JiIiwiZX
hwIjoxNDg0MTI5NzczLCJub21icmUiOiJhZG9sZm9DYW1iaW9zIiwiY29udHJhc2XDsWEiOiJhZG9sZm9DYW1iaW9zIiwiaWQiOjg0fQ.m0P
PstEZrax0i-NHN3k9Gr82Br9TbzTr5PeFdSMNn3g

 -- response --
200 OK
Server:  Apache-Coyote/1.1
Authorization:
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwianRpIjoiNmMyNDRlNTItODNlOS00MDY5LWFhMDMtNDIyYWE4Yjk4N2JiIiwiZX
hwIjoxNDg0MTI5ODM0LCJub21icmUiOiJhZG9sZm9DYW1iaW9zIiwiY29udHJhc2XDsWEiOiJhZG9sZm9DYW1iaW9zIiwiaWQiOjg0fQ.1-
bFgapL22DKyxEzM1WKx3QPD4gMfORj2wi2sf1gF-g
Content-Type:  application/json
Transfer-Encoding:  chunked
Date:  Wed, 11 Jan 2017 10:07:14 GMT

{"bibliotecas":[{"descripcion":"Libros de casa de Madrid","idBiblioteca":"63","idUsuario":"84","nombreBiblioteca":"Libros
modificado","numBibliotecas":"0"},{"descripcion":"libros secundarios","idBiblioteca":"64","idUsuario":"84","nombreBiblioteca":"biblioteca
2","numBibliotecas":"0"}]}
```


Figura 55. Prueba llamada todas las bibliotecas

6.1.7. Obtener información de una biblioteca

Con esta prueba se verifica que el servicio devuelve la información correspondiente a una biblioteca determinada. El servicio recibe el identificador de la biblioteca en la URL que invoca al método necesario. Como podemos ver en la figura 56, el servicio realiza la búsqueda de manera correcta devolviendo los datos de la biblioteca indicada, así como un código de respuesta correcto, 200.


```
GET http://localhost:8080/gestionBibliotecas/bibliotecas/biblioteca/63
Authorization:
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwianRpIjoiNmMyNDRlNTItODNlOS00MDY5LWFhMDMtNDIyYWE4Yjk4N2JiIiwiZX
hwIjoxNDg0MTI5NzczLCJub21icmUiOiJhZG9sZm9DYW1iaW9zIiwiY29udHJhc2XDsWEiOiJhZG9sZm9DYW1iaW9zIiwiaWQiOjg0fQ.m0P
PstEZrax0i-NHN3k9Gr82Br9TbzTr5PeFdSMNn3g

 -- response --
200 OK
Server:  Apache-Coyote/1.1
Content-Type:  application/json
Transfer-Encoding:  chunked
Date:  Wed, 11 Jan 2017 10:20:06 GMT

{"descripcion":"Libros de casa de Madrid","idBiblioteca":"63","idUsuario":"84","nombreBiblioteca":"Libros modificado","numBibliotecas":"0"}
```


Figura 56. Prueba llamada información de una biblioteca

6.1.8. Crear una nueva biblioteca

En esta prueba el servicio recibe como parámetros la información de una biblioteca: nombre y descripción. Se comprueba que la creación de una nueva biblioteca es correcta y la respuesta del servicio también lo es. Podemos ver en la figura 57 cómo la respuesta del servicio es un 201 indicando que la operación se ha realizado satisfactoriamente. A su vez, en la figura 58 podemos corroborar la creación de la misma en la base de datos.


```
POST http://localhost:8080/gestionBibliotecas/bibliotecas/biblioteca?nombre=Biblioteca%20prueba&descripcion=pruebas
Authorization:
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwianRpIjoiNmMyNDRlNTItODNlOS00MDY5LWFhMDMtNDIyYWE4Yjk4N2JiIiwiZX
hwIjoxNDg0MTMxMjE1LCJub21icmUiOiJhZG9sZm9DYW1iaW9zIiwiY29udHJhc2XDsWEiOiJhZG9sZm9DYW1iaW9zIiwiaWQiOjg0fQ.2p
WHH29xB-rAVQFR8IrJ0nckUCcOXI-GnGT1VS6vnrk
Content-Type: application/x-www-form-urlencoded

 -- response --
201 Creado
Server:  Apache-Coyote/1.1
Authorization:
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwianRpIjoiNmMyNDRlNTItODNlOS00MDY5LWFhMDMtNDIyYWE4Yjk4N2JiIiwiZX
hwIjoxNDg0MTMxMjYyLCJub21icmUiOiJhZG9sZm9DYW1iaW9zIiwiY29udHJhc2XDsWEiOiJhZG9sZm9DYW1iaW9zIiwiaWQiOjg0fQ.Np
A-O_O37SSjxGSavdGCUyxVhVSbHSes3woLdNXXFxc
Content-Type:  application/json
Transfer-Encoding:  chunked
Date:  Wed, 11 Jan 2017 10:31:02 GMT

Biblioteca creada satisfactoriamente
```


Figura 57. Prueba llamada crear una biblioteca

idBibliotecas	idUsuario	nombre	descripcion
65	84	Biblioteca prueba	pruebas

Figura 58. Resultado base de datos creación de biblioteca

6.1.9. Actualizar la información de una biblioteca

El objetivo es comprobar que la información de una biblioteca es modificada correctamente con los nuevos datos enviados. El servicio recibe la información modificada de una biblioteca determinada, nombre y descripción. A través de la URL, el servicio también recibe el id de la biblioteca. En la figura 59 se puede visualizar que el servicio devuelve un mensaje correcto al realizarse la operación de manera satisfactoria. Para comprobar el cambio en la figura 58 visualizamos el estado anterior de la biblioteca y en la figura 60 podemos visualizar el estado final de la misma.


```
POST http://localhost:8080/gestionBibliotecas/bibliotecas/biblioteca/65?nombre=Biblioteca%20prueba%20cambiada&
descripcion=pruebas%20cambiadas
Authorization:
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwianRpIjoiNmMyNDRlNTItODNlOS00MDY5LWFhMDMtNDIyYWE4Yjk4N2JiIiwiZX
hwIjoxNDg0MTMxMjE1LCJub21icmUiOiJhZG9sZm9DYW1iaW9zIiwiY29udHJhc2XDsWEiOiJhZG9sZm9DYW1iaW9zIiwiaWQiOjg0fQ.2p
WHH29xB-rAVQFR8IrJ0nckUCcOXI-GnGT1VS6vnrk
Content-Type: application/x-www-form-urlencoded

 -- response --
200 OK
Server:  Apache-Coyote/1.1
Content-Type:  application/json
Transfer-Encoding:  chunked
Date:  Wed, 11 Jan 2017 11:23:54 GMT

La informacion de la biblioteca se ha modificado correctamente.
```


Figura 59. Prueba llamada editado biblioteca

idBibliotecas	idUsuario	nombre	descripcion
65	84	Biblioteca prueba cambiada	pruebas cambiadas

Figura 60. Resultado base de datos editado biblioteca

6.1.10. Eliminar una biblioteca del sistema

En esta prueba invocamos al método de eliminar una biblioteca del servicio el cual recibe el identificador de la misma. Al producirse la operación de manera correcta el servicio envía un código 200 y el resultado de la operación como podemos comprobar en la figura 61. A su vez, en la figura 62 podemos comprobar que se han eliminado tanto la biblioteca indicada (id=64) y todos sus libros.


```
DELETE http://localhost:8080/gestionBibliotecas/bibliotecas/bibliotec
Authorization:
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwianRpIjoiNm
hwIjoxNDg0MTMxMjE1LCJub21icmUiOiJhZG9sZm9DYW1iaW9zIiw
WHH29xB-rAVQFR8IrJ0nckUCcOXI-GnGT1VS6vnrk

 -- response --
200 OK
Server: Apache-Coyote/1.1
Content-Type: application/json
Transfer-Encoding: chunked
Date: Wed, 11 Jan 2017 11:36:50 GMT

Recurso eliminado correctamente
```


Figura 61. Prueba llamada eliminar biblioteca

idBibliotecas	idUsuario
58	76
59	68
60	81
61	81
62	82
63	84
65	84
NULL	NULL

idLibros	idBiblioteca
36	59
42	59
95	62
96	62
97	62
98	62
99	63
NULL	NULL

Figura 62. Resultado base de datos eliminar biblioteca

6.1.11. Obtener todos los libros de una biblioteca

El servicio recibe un identificador de una biblioteca y devuelve correctamente un mensaje con un código 200 y un fichero JSON con la información de todos los libros de esa biblioteca como podemos corroborar en la figura 63. Se ha comprobado también que en caso de que no haya libros se devuelve NULL y un error en la consulta devuelve un código de respuesta 500.


```
GET http://localhost:8080/gestionBibliotecas/bibliotecas/miBiblioteca/63/libros
Authorization:
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwianRpIjoiNmMyNDRlNTItODNlOS00MDY5LWFhMDMtNDIyYWE4Yjk4N2JiIiwiZX
hwIjoxNDg0MTMxMjE1LCJub21icmUiOiJhZG9sZm9DYW1iaW9zIiwiY29udHJhc2XDsWEiOiJhZG9sZm9DYW1iaW9zIiwiaWQiOjg0fQ.2p
WHH29xB-rAVQFR8IrJ0nckUCcOXI-GnGT1VS6vnrk

 -- response --
200 OK
Server: Apache-Coyote/1.1
Content-Type: application/json
Transfer-Encoding: chunked
Date: Wed, 11 Jan 2017 11:52:23 GMT

{"libros":[{"autor":"Tanembaum","columna":"0","editorial":"Pearson
Education","fila":"0","genero":"Informatica","ISBN":"0-13-088893-1","id":"99","idBiblioteca":"63","imagen":"","numLibros":"0","paginas":"803"
,"resumen":"","saga":"","titulo":"Distributed system"},
{"autor":"adolfo","columna":"0","editorial":"NA","fila":"0","genero":"info","ISBN":"","id":"100","idBiblioteca":"63","imagen":"","numLibros":"0","
paginas":"0","resumen":"","saga":"","titulo":"libro1"},
{"autor":"adolfo","columna":"0","editorial":"N/A","fila":"0","genero":"info","ISBN":"","id":"101","idBiblioteca":"63","imagen":"","numLibros":"0","
paginas":"0","resumen":"","saga":"","titulo":"Libro2"},{"autor":"adolfo","columna":"0","editorial":"N
/A","fila":"0","genero":"info","ISBN":"","id":"102","idBiblioteca":"63","imagen":"","numLibros":"0","paginas":"0","resumen":"","saga":"","titulo":"
Libro3"}]}
```


Figura 63. Prueba llamada obtener todos los libros

6.1.12. Buscar un libro por id y por nombre

El servicio da la posibilidad de poder buscar un libro tanto por su identificador como por su nombre. En las figura 64 y 65 se comprueba que la búsqueda es correcta y se envía un código de operación 200 junto con un fichero JSON con el resultado de la operación, la información del libro buscado.


```
GET http://localhost:8080/gestionBibliotecas/bibliotecas/miBiblioteca
/63/libros/101
Authorization:
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwianRpIjoiNmMyN
DRlNTItODNlOS00MDY5LWFhMDMtNDIyYWE4Yjk4N2JiIiwiZXhwIjoxND
g0MTMxMjE1LCJub21icmUiOiJhZG9sZm9DYW1iaW9zIiwiY29udHJhc2X
DsWEiOiJhZG9sZm9DYW1iaW9zIiwiaWQiOjg0fQ.2pWHH29xB-
rAVQFR8IrJ0nckUCcOXI-GnGT1VS6vnrk

 -- response --
200 OK
Server:  Apache-Coyote/1.1
Content-Type:  application/json
Transfer-Encoding:  chunked
Date:  Wed, 11 Jan 2017 12:04:30 GMT

{"autor":"adolfo","columna":"0","editorial":"N/A","fila":"0","genero":"info","IS
BN":"","id":"101","idBiblioteca":"63","imagen":"","numLibros":"0","paginas":"
0","resumen":"","saga":"","titulo":"Libro2"}
```


Figura 64. Prueba búsqueda libro por id


```
GET http://localhost:8080/gestionBibliotecas/bibliotecas/miBiblioteca
/63/libros/nombre/Libro2
Authorization:
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwianRpIjoiNmMyN
DRlNTItODNlOS00MDY5LWFhMDMtNDIyYWE4Yjk4N2JiIiwiZXhwIjoxND
g0MTMxMjE1LCJub21icmUiOiJhZG9sZm9DYW1iaW9zIiwiY29udHJhc2X
DsWEiOiJhZG9sZm9DYW1iaW9zIiwiaWQiOjg0fQ.2pWHH29xB-
rAVQFR8IrJ0nckUCcOXI-GnGT1VS6vnrk

 -- response --
200 OK
Server:  Apache-Coyote/1.1
Content-Type:  application/json
Transfer-Encoding:  chunked
Date:  Wed, 11 Jan 2017 12:03:09 GMT

{"libros":
{"autor":"adolfo","columna":"0","editorial":"N/A","fila":"0","genero":"info","IS
BN":"","id":"101","idBiblioteca":"63","imagen":"","numLibros":"0","paginas":"
0","resumen":"","saga":"","titulo":"Libro2"}}
```


Figura 65. Prueba búsqueda libro por nombre

6.1.13. Crear un libro en una biblioteca

En esta prueba comprobamos que el servicio, recibiendo los parámetros obligatorios de creación de un libro lo añade a la biblioteca correctamente. El identificador de la biblioteca se añade en la URL que apunta al servicio. En la figura 66 se ve como el mensaje de respuesta generado por el servicio es correcto con un código 201 y mensaje de la operación. En la figura 67 se comprueba la creación satisfactoria del libro en la base de datos.


```
POST http://localhost:8080/gestionBibliotecas/bibliotecas/miBiblioteca
/63/libros?titulo=LibroPrueba&autor=adolfo&genero=accion&
editorial=N%2FA
Authorization:
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwianRpIjoiNmMyN
DRlNTItODNlOS00MDY5LWFhMDMtNDIyYWE4Yjk4N2JiIiwiZXhwIjoxND
g0MTMxMjE1LCJub21icmUiOiJhZG9sZm9DYW1iaW9zIiwiY29udHJhc2X
DsWEiOiJhZG9sZm9DYW1iaW9zIiwiaWQiOjg0fQ.2pWHH29xB-
rAVQFR8IrJ0nckUCcOXI-GnGT1VS6vnrk
Content-Type: application/x-www-form-urlencoded

 -- response --
201 Creado
Server:  Apache-Coyote/1.1
Content-Type:  application/xml
Transfer-Encoding:  chunked
Date:  Wed, 11 Jan 2017 12:14:34 GMT

Libro creado correctamente
```


Figura 66. Prueba respuesta creación libro

idLibros	idBiblioteca	titulo	autor	genero	saga	editorial	paginas	ISBN	sinopsis	fila	columna
106	63	LibroPrueba	adolfo	accion	NULL	N/A	0	NULL	NULL	0	0

Figura 67. Prueba base de datos creación libro

6.1.14. Actualizar información de un libro

En esta prueba el servicio recibe la información obligatoria para la creación de un libro y el identificador del libro a modificar. En la figura 68 se comprueba la respuesta enviada por el servicio ante una operación satisfactoria, código 200 y resultado de la operación. En la figura 69 se puede comprobar como la información del libro ha sido modificada con éxito en la base de datos.


```
POST http://localhost:8080/gestionBibliotecas/bibliotecas/miBiblioteca
/63/libros/101?titulo=LibroPruebaModificado&autor=adolfoModificado&
genero=accionModificado&editorial=N%2FA
Authorization:
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwianRpIjoiNmMyN
DRlNTItODNlOS00MDY5LWFhMDMtNDIyYWE4Yjk4N2JiIiwiZXhwIjoxND
g0MTMxMjE1LCJub21icmUiOiJhZG9sZm9DYW1iaW9zIiwiY29udHJhc2X
DsWEiOiJhZG9sZm9DYW1iaW9zIiwiaWQiOjg0fQ.2pWHH29xB-
rAVQFR8IrJ0nckUCcOXI-GnGT1VS6vnrk
Content-Type: application/x-www-form-urlencoded

 -- response --
200 OK
Server:  Apache-Coyote/1.1
Content-Type:  application/xml
Transfer-Encoding:  chunked
Date:  Wed, 11 Jan 2017 12:21:10 GMT

La información del libro ha sido modificada.
```


Figura 68. Prueba respuesta editar libro

idLibros	idBiblioteca	titulo	autor	genero	saga	editorial
101	63	LibroPruebaModi...	adolfoModi...	accionMo...	NULL	N/A

Figura 69. Prueba base de datos editar libro

6.1.15. Eliminar un libro de una biblioteca

El servicio recibe un identificador de un libro y procede a eliminar ese recurso de la base de datos. En la figura 70 se comprueba el mensaje de respuesta del servicio ante una operación realizada correctamente con un código 200 y el mensaje de la operación. A su vez, en la figura 71 se visualiza cómo el libro ha sido eliminado satisfactoriamente de la base de datos.


```
DELETE http://localhost:8080/gestionBibliotecas/bibliotecas/miBiblioteca
/63/libros/101
Authorization:
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwianRpIjoiNmMyN
DRlNTItODNlOS00MDY5LWFhMDMtNDIyYWE4Yjk4N2JiIiwiZXhwIjoxND
g0MTMxMjE1LCJub21icmUiOiJhZG9sZm9DYW1iaW9zIiwiY29udHJhc2X
DsWEiOiJhZG9sZm9DYW1iaW9zIiwiaWQiOjg0fQ.2pWHH29xB-
rAVQFR8IrJ0nckUCcOXI-GnGT1VS6vnrk

 -- response --
200 OK
Server:  Apache-Coyote/1.1
Content-Type:  text/plain
Transfer-Encoding:  chunked
Date:  Wed, 11 Jan 2017 12:23:04 GMT

Libro eliminado correctamente
```


Figura 70. Prueba respuesta eliminar libro

idLibros	idBiblioteca
95	62
96	62
97	62
98	62
99	63
100	63
102	63
103	63
104	63
105	63
106	63
NULL	NULL

Figura 71. Prueba base de datos eliminar libro

6.1.16. Validez del token recibido

En cada petición que se envía al servicio en la cabecera HTTP *Authoritation* debe ir el *token* de sesión generado. El servicio antes de realizar el método requerido, comprueba la validez del *token* recibido y si concuerda con el usuario conectado. En la figura 72 podemos visualizar la respuesta del servicio en caso de que el *token* no sea correcto o haya caducado, en este caso el método no se ejecuta. Esto mismo ocurre en todos los métodos implementados del servicio y comprobados en los casos anteriores. Esta prueba se realiza a nivel general para comprobar que el servicio hace una comprobación correcta del *token* recibido.

```
POST http://localhost:8080/gestionBibliotecas/bibliotecas/usuario?usaurio=adolf&contrasenia=adolf&email=adolf%40gmail.com
Content-Type: application/x-www-form-urlencoded

 -- response --
412 Precondición Fallada
Server:  Apache-Coyote/1.1
Content-Type:  application/xml
Content-Length:  0
Date:  Tue, 10 Jan 2017 16:05:00 GMT
```

Figura 72. Prueba detección token erróneo

6.1.17. Mensaje de error generado por error en el método HTTP enviado

Siempre que el servicio recibe una petición determinada y el método HTTP enviado asociado a una URL no está permitido, el servidor responde con un mensaje de error. El funcionamiento es correcto como se puede observar en la figura 73.

```
PUT http://localhost:8080/gestionBibliotecas/bibliotecas/miBiblioteca
/63/libros/101
Content-Type: application/json

 -- response --
405 Método No Permitido
Server:  Apache-Coyote/1.1
Allow:  HEAD,DELETE,POST,GET,OPTIONS
Content-Length:  0
Date:  Wed, 11 Jan 2017 12:37:32 GMT
```

Figura 73. Prueba respuesta por error en el método

6.2. Verificación del cliente web

6.2.1. Acceso a la aplicación

En esta primera prueba comprobamos el correcto funcionamiento del acceso a la aplicación. En la figura 74 se observa cómo, tras introducir un usuario y contraseña incorrectos, el sistema nos muestra un mensaje de error. Si en cambio el acceso ha sido correcto, accedemos directamente a la ventana de selección de biblioteca como se observa en la figura 75.

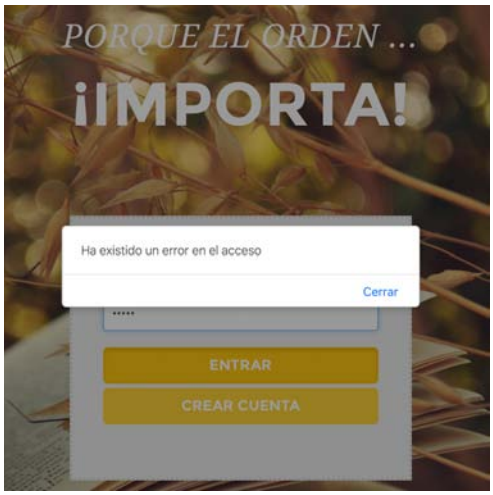


Figura 74. Prueba interfaz error acceso

Figura 75. Prueba interfaz selección bibliotecas

6.2.2. Creación de un nuevo usuario

En esta prueba, el objetivo es verificar que el formulario valida las entradas y el registro de un usuario se realiza correctamente.

En las figuras 76 y 77 comprobamos que efectivamente no podemos realizar un registro si no están todos los campos rellenados y además que se haya introducido un formato de email correcto.

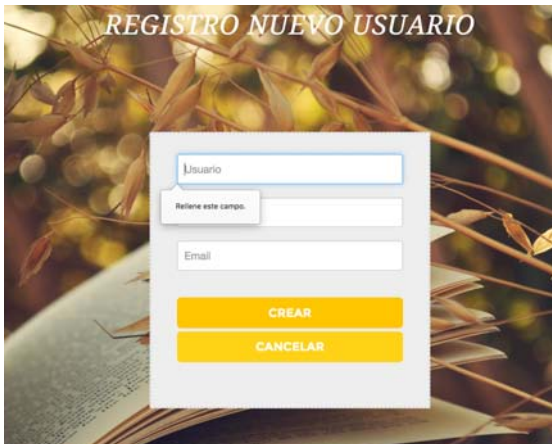


Figura 76. Prueba interfaz campos formulario

Figura 77. Prueba interfaz campo email formulario

En las imágenes 78 y 79 se verifica que los mensajes mostrados por el sistema como resultado de la operación son correctos tanto si el registro se ha realizado satisfactoriamente como si ha sufrido algún error.

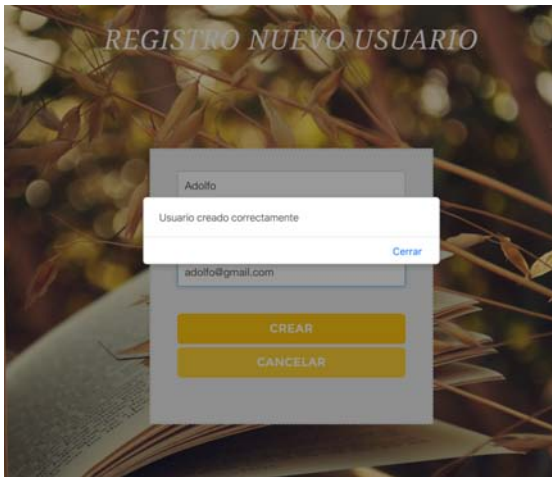


Figura 78. Prueba interfaz registro correcto

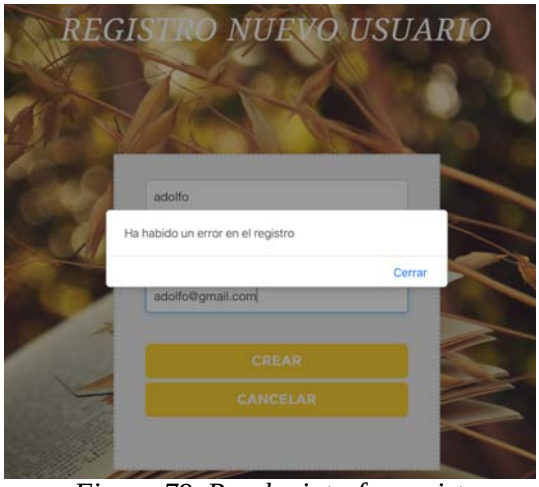


Figura 79. Prueba interfaz registro erróneo

6.2.3. Selección de bibliotecas

El objetivo de comprobar la selección de bibliotecas es que como podemos observar en la figura 80, si el usuario no ha creado ninguna biblioteca con anterioridad, no se le permite realizar ninguna operación salvo la de crear una biblioteca. En cambio, como se visualiza en la figura 81, el usuario tiene una biblioteca creada en el sistema y puede así operar con cualquiera de las operaciones que existen.

Figura 80. Prueba interfaz no hay bibliotecas

Figura 81. Prueba interfaz muestra de bibliotecas

6.2.4. Editar biblioteca

En la siguiente prueba se comprueba que en el formulario de edición se obtienen de manera correcta los datos relacionados con la biblioteca seleccionada como se observa en la figura 83.
Tras realizar la edición el sistema vuelve automáticamente al menú de selección de bibliotecas donde podemos observar el cambio realizado en la lista de bibliotecas comparando la figura 84 y la figura 82.

Figura 82. Prueba formulario edición de biblioteca

Figura 83. Prueba interfaz selección de bibliotecas con cambios

6.2.5. Menú de inicio de la biblioteca

En la figura 85 se observa el acceso a la biblioteca seleccionada en el paso anterior y cómo se visualizan de manera correcta algunos de los libros de esa biblioteca.

Figura 84. Prueba interfaz inicio

6.2.6. Búsqueda de un libro

En la siguiente prueba comprobamos que el cliente procesa correctamente la información devuelta por el servicio Web respecto a la búsqueda de un libro. En la figura 86 vemos cómo la aplicación muestra un resultado diferente en el caso de que no se encuentre el libro. En las figuras 87 y 88 se observa el resultado del libro y la visualización de su contenido correctamente.

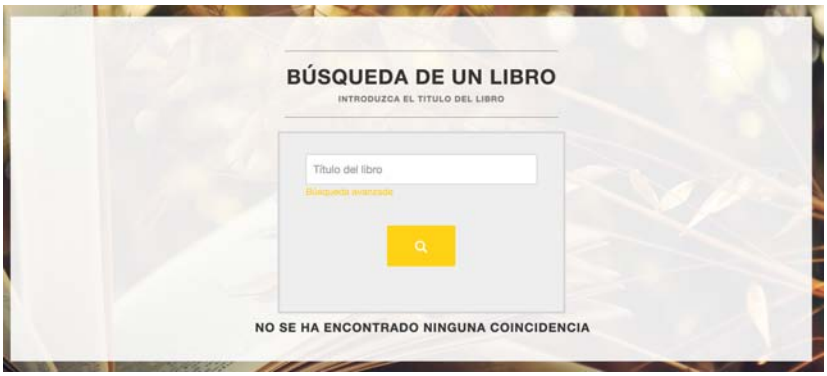


Figura 85. Prueba interfaz búsqueda sin resultados

Figura 86. Prueba interfaz resultado búsqueda

Figura 87. Prueba información libro encontrado

6.2.7. Formulario crear un nuevo libro

En esta comprobación vemos que de manera correcta, el formulario valida las entradas y no se permite enviar información en caso de error en alguna de las entradas obligatorias como se muestra en la figura 88. En la figura 89 se ve el mensaje del resultado de la operación generado por el sistema.

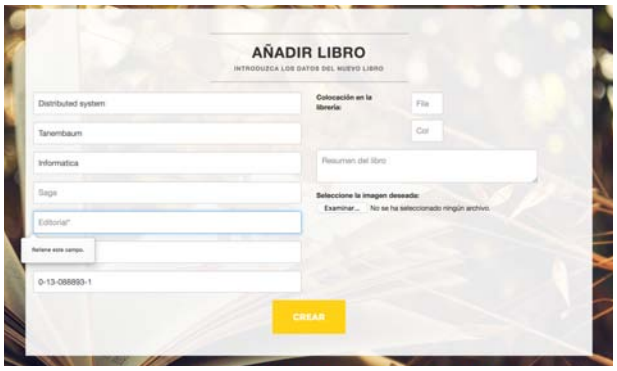


Figura 88. Prueba interfaz campos del formulario crear libro

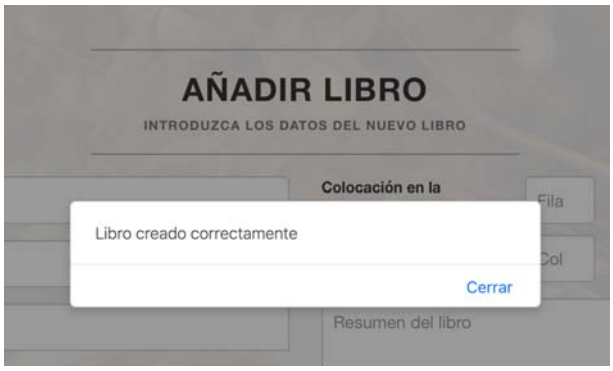
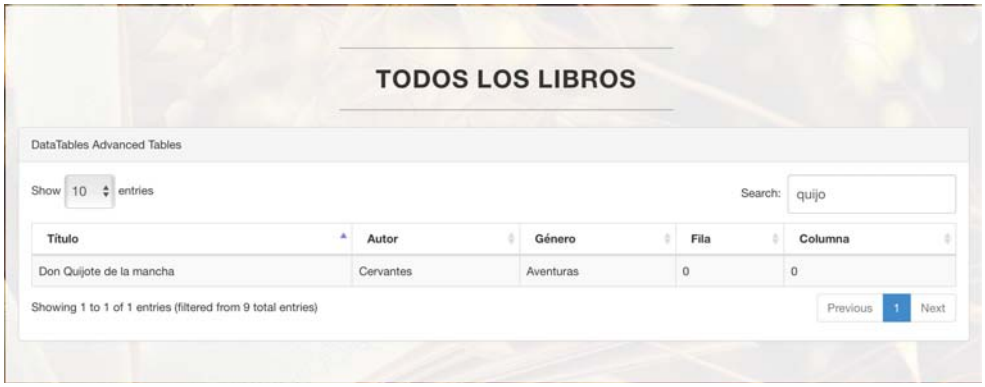


Figura 89. Prueba mensaje creación libro

6.2.8. Interfaz visualización todos los libros

La prueba siguiente confirma que se visualiza de manera correcta todos los libros añadidos a la biblioteca, reflejados en la figura 90. A su vez, se ha comprobado el correcto funcionamiento de la función buscar de manera satisfactoria, como muestra la figura 91.

Figura 90. Prueba tabla todos los libros



Prueba 91. Prueba búsqueda en tabla todos

6.2.9. Editar un libro de la biblioteca

El objetivo de esta prueba es comprobar que a la hora de editar la información de un libro, obtenemos de manera correcta todos los datos insertados con anterioridad. Como se ve en la figura 92 se visualizan de manera correcta todos los datos del libro encontrado. También se ha podido comprobar la validación del formulario así como el correcto cambio de los nuevos datos insertados.

Figura 92. Prueba formulario de edición de libros

6.2.10. Editar perfil

Al igual que en la prueba anterior se observa en la figura 93 que de manera correcta se obtienen los datos del usuario conectado. También se ha verificado la correcta validación del formulario y edición de los nuevos parámetros introducidos.

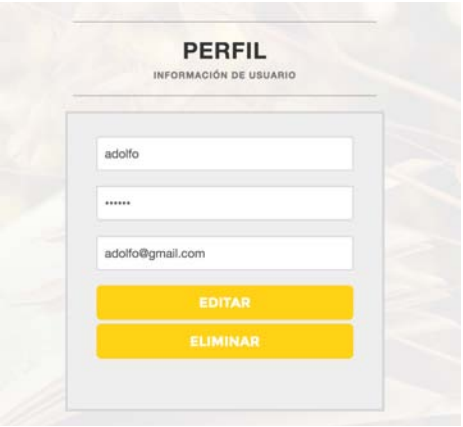


Figura 93. Prueba formulario del perfil

6.2.11. Salir de la aplicación

Se ha comprobado de manera satisfactoria que tras cerrar sesión, se pierden todas las variables de sesión creadas por el cliente y no se puede realizar ninguna operación sin volver a registrarse.

6.2.12. Caducidad del token

El fin de la prueba es constatar que, transcurrido el periodo de expiración asociado al *token* el sistema no permite realizar ninguna operación, mostrando una operación en pantalla como se visualiza en la figura 94. Tras cerrar el mensaje de aviso el propio sistema abre el formulario de login para un nuevo acceso de manera correcta.

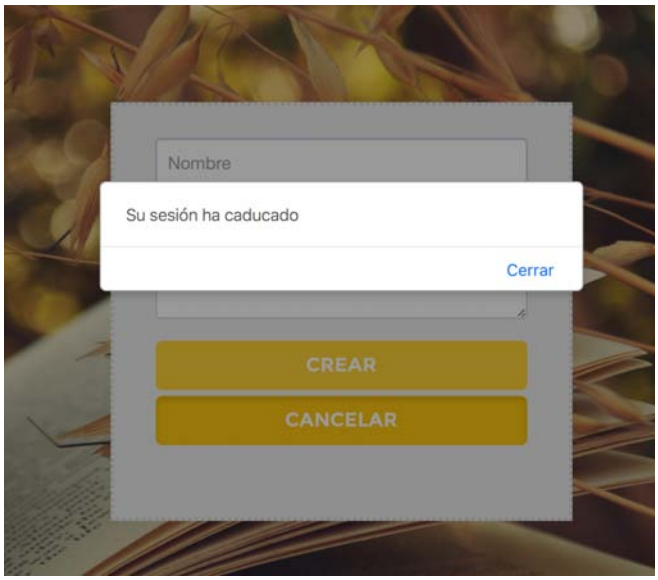


Figura 94. Prueba mensaje del cliente por caducidad del token de sesión

6.2.13. Mensaje de confirmación al eliminar un recurso

Para finalizar con las pruebas del cliente, en la figura 95 corroboramos cómo el sistema muestra un mensaje de comprobación a la hora de eliminar cualquier recurso.

Figura 95. Prueba mensaje de confirmación

7- Conclusiones y lineas a mejorar

El desarrollo de este proyecto me ha permitido conocer de primera mano una tecnología que está emergiendo en los últimos años y se espera que sea una de las opciones de desarrollo de servicios Web más utilizadas en el futuro.

A su vez, me ha permitido aprender cómo diseñar una aplicación desde cero siguiendo una metodología de ingeniería del software profundizando en conceptos como SRS, diagramas de caso de uso, diagrama de actividades, modelado del diagrama de clases y de implementación, entre otras muchas cosas.

Las aptitudes aprendidas o mejoradas no solo se centran en el estudio e implementación de esta nueva tecnología sino al realizar este proyecto me ha permitido conocer otros muchos campos como pueden ser las bases de datos, administración de sistemas operativos, configuración de balanceadores de carga, etc

Cabe destacar la capacidad de organización adquirida gracias a las diferentes fases desarrolladas durante todo el proyecto.

A pesar de todo lo desarrollado, se ha realizado un proyecto que puede mejorar su funcionalidad en varios aspectos. Además los futuros cambios a realizar no implicarían modificar nada de lo implementado gracias al diseño realizado del servicio REST, únicamente se añadiría más funcionalidad.

Algunas de las posibles mejoraras en un futuro, podrían ser:

- Conexión directa con otros Web Services para ampliar automáticamente la información de los libros añadidos

- Una actualización interesante sería permitir la subida de *ebooks* asociados al libro añadido. De esta manera se podría descargar y tener accesible en cualquier momento y desde cualquier plataforma cualquiera de los libros

- Otras funciones a implementar serían la interacción con bibliotecas municipales mediante llamadas REST añadiéndole una sección de préstamo de libros

- El objetivo de realizar un servicio Web es, a la vez, una gran opción para utilizarse en aplicaciones multi-plataforma por lo que la creación de nuevos clientes que consuman el servicio es fundamental para aprovechar al máximo el servicio

BIBLIOGRAFÍA

A. Fernández (2013, 11). *Servicios Web Restful con HTTP. Parte 1: Introducción y beneficios*. Disponible en: http://www.adwe.es/general/colaboraciones/servicios-web-restful-con-http-parte-i-introduccion-y-bases-teoricas

A. Sha (2016, 11). How to build RESTful Service with Java using JAX-RS and Jersey (Example). Disponible en: http://crunchify.com/how-to-build-restful-service-with-java-using-jax-rs-and-jersey/

Apache Maven Project. *What is Maven?* Disponible en: http://maven.apache.org/index.html

A. S. Tanenbaum and M. V. Steen, *Distributed Systems. Principles and Paradigms.* New Yersey: Prentice Hall, 2002.

B. Burke, *RESTful Java with JAX-RS 2.0.* Sebastopol: O'Reilly: 2013

C. Álvarez Caules (2013, 06). *Introducción a Servicios REST*. Disponible en: http://www.arquitecturajava.com/servicios-rest/

C. Álvarez (2014,08). *¿Qué es Maven?*. Disponible en: https://www.genbetadev.com/autor/cecilio

D. Gourley and Brian Totty, *HTTP. The Definitive Guide.* Sebastopol: O'Reilly: 2002

D. Silva (2010, 11). *RESTful… la forma más ligera de hacer WebServices (Parte 1)*. Apuntes de Java. Disponible en: http://www.apuntesdejava.com/2010/11/restful-la-forma-mas-ligera-de-hacer.html

E. Amodeo (2010, 07). *¿Qué es REST?*. Te lo dije… Ingeniería del Software en el siglo XXI. Disponible en: https://eamodeorubio.wordpress.com/2010/07/26/servicios-web-2-¿que-es-rest/

G. Jiménez Centeno (2008, 4). *Jersey: la implementación de RESTFull de Sun*. Disponible en: https://www.adictosaltrabajo.com/tutoriales/jersey-rest/

J. Guerra (2013, 07). *Vídeo Sistemas Distribuidos (Proyecto Web Service REST for Android)*. Disponible en: https://www.youtube.com/watch?v=XRURZVtphn8

Java. *The Java EE 6 Tutorial*. Disponible en: http://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html

Java Brains. (2015). *Developing RESTful APIs with JAX-RS*. Disponible en: https://www.youtube.com/playlist?list=PLqq-6Pq4lTTZh5U8RbdXq0WaYvZBz2rbn

Java Brains (2016). *Advanced JAX-RS*. Disponible en: https://www.youtube.com/playlist?list=PLqq-6Pq4lTTY40IcG584ynNqibMc1heIa

Jesin A (2002,12). *How to use HAPorxy to set up MySQL load balancing.* Disponible en:

https://www.digitalocean.com/community/tutorials/how-to-use-haproxy-to-set-up-mysql-load-balancing--3#testing-load-balancing-and-failover

Jersey. *RESTful Web Services in Java*. Disponible en:
https://jersey.java.net

Josue (2010, 11). *REST(full) web services*. Disponible en:
http://permitanquemepresente.blogspot.com.es/2010/11/restfull-web-services.html

L. Richardson and S. Ruby, *RESTful Web Services.* Sebastopol: O'Reilly, 2007.

L. de Seta (2008, 11). *Introducción a los servicios web RESTful*. Dos Ideas, personas y software. Disponible en:
http://www.dosideas.com/noticias/java/314-introduccion-a-los-servicios-web-restful.html

L. Vogel (2009). *REST with Java (JAX-RS) using Jersey - Tutorial*. Disponible en:
http://www.vogella.com/tutorials/REST/article.html

M. Debnath. (2013,11). *Creating RESTful Web Services with JAX-RS*. Disponible:
http://www.developer.com/java/creating-restful-web-services-with-jax-rs.html

M. Grinberg (2013, 06). *Writing a Javascript REST client*. Disponible en:
https://blog.miguelgrinberg.com/post/writing-a-javascript-rest-client

M. G. Piattini, J. A. Calvo-Manzano, J. Cervera and L. Fernández, *Análisis y diseño de Aplicaciones Informáticas de Gestión. Una perspectiva de Ingeniería del Software.* Madrid: Ra-Ma, 2003.

M. Salías (2011). *Vídeo arquitectura REST*. Disponible en:
https://www.youtube.com/watch?v=CBjUCyt6Z8E

N. Sakimura (2015, 05). *JASON Web Token (JWT).* Disponible en:
http://self-issued.info/docs/draft-ietf-oauth-json-web-token.html#rfc.section.7

T. Tkalec (2010). JSON Web Token Tutorial: An Example in Laravel and Angular JS. Disponible:
https://www.toptal.com/web/cookie-free-authentication-with-json-web-tokens-an-example-in-laravel-and-angularjs

Unknown (2014, 06). *Java EE / JAX-RS 2.0: Simple REST API Authentication & Authorization with Custom HTTP header.* Disponible en:
http://www.developerscrappad.com/1814/java/java-ee/rest-jax-rs/java-ee-7-jax-rs-2-0-simple-rest-api-authentication-authorization-with-custom-http-header/

ANEXOS

a) Conversación de Ryan Tomayko con su esposa

Wife: Who is Roy Fielding?

Ryan: Some guy. He's smart.

Wife: Oh? What did he do?

Ryan: He helped write the first web servers and then did a ton of research explaining why the web works the way it does. His name is on the specification for the protocol that is used to get pages from servers to your browser.

Wife: How does it work?

Ryan: The web?

Wife: Yeah.

Ryan: Hmm. Well, it's all pretty amazing really. And the funny thing is that it's all very undervalued. The protocol I was talking about, HTTP, it's capable of all sorts of neat stuff that people ignore for some reason.

Wife: You mean http like the beginning of what I type into the browser?

Ryan: Yeah. That first part tells the browser what protocol to use. That stuff you type in there is one of the most important breakthroughs in the history of computing.

Wife: Why?

Ryan: Because it is capable of describing the location of something anywhere in the world from anywhere in the world. It's the foundation of the web. You can think of it like GPS coordinates for knowledge and information.

Wife: For web pages?

Ryan: For anything really. That guy, Roy Fielding, he talks a lot about what those things point to in that research I was talking about. The web is built on an architectural style called REST. REST provides a definition of a resource, which is what those things point to.

Wife: A web page is a resource?

Ryan: Kind of. A web page is a representation of a resource. Resources are just concepts. URLs--those things that you type into the browser...

Wife: I know what a URL is..

Ryan: Oh, right. Those tell the browser that there's a concept somewhere. A browser can then go ask for a specific representation of the concept. Specifically, the browser asks for the web page representation of the concept.

Wife: What other kinds of representations are there?

Ryan: Actually, representations is one of these things that doesn't get used a lot. In most cases, a resource has only a single representation. But we're hoping that representations will be used more in the future because there's a bunch of new formats popping up all over the place.

Wife: Like what?

Ryan: Hmm. Well, there's this concept that people are calling Web Services. It means a lot of different things to a lot of different people but the basic concept is that machines could use the web just like people do.

Wife: Is this another robot thing?

Ryan: No, not really. I don't mean that machines will be sitting down at the desk and browsing the web. But computers can use those same protocols to send messages back and forth to each other. We've been doing that for a long time but none of the techniques we use today work well when you need to be able to talk to all of the machines in the entire world.

Wife: Why not?

Ryan: Because they weren't designed to be used like that. When Fielding and his buddies started building the web, being able to talk to any machine anywhere in the world was a primary concern. Most of the techniques we use at work to get computers to talk to each other didn't have those requirements. You just needed to talk to a small group of machines.

Wife: And now you need to talk to all the machines?

Ryan: Yes - and more. We need to be able to talk to all machines about all the stuff that's on all the other machines. So we need some way of having one machine tell another machine about a resource that might be on yet another machine.

Wife: What?

Ryan: Let's say you're talking to your sister and she wants to borrow the sweeper or something. But you don't have it - your Mom has it. So you tell your sister to get it from your Mom instead. This happens all the time in real life and it happens all the time when machines start talking too.

Wife: So how do the machines tell each other where things are?

Ryan: The URL, of course. If everything that machines need to talk about has a corresponding URL, you've created the machine equivalent of a noun. That you and I and the rest of the world have agreed on talking about nouns in a certain way is pretty important, eh?

Wife: Yeah.

Ryan: Machines don't have a universal noun - that's why they suck. Every programming language, database, or other kind of system has a different way of talking about nouns. That's why the URL is so important. It let's all of these systems tell each other about each other's nouns.

Wife: But when I'm looking at a web page, I don't think of it like that.

Ryan: Nobody does. Except Fielding and handful of other people. That's why machines still suck.

Wife: What about verbs and pronouns and adjectives?

Ryan: Funny you asked because that's another big aspect of REST. Well, verbs are anyway.

Wife: I was just joking.

Ryan: It was a funny joke but it's actually not a joke at all. Verbs are important. There's a powerful concept in programming and CS theory called polymorphism. That's a geeky way of saying that different nouns can have the same verb applied to them.

Wife: I don't get it.

Ryan: Well.. Look at the coffee table. What are the nouns? Cup, tray, newspaper, remote. Now, what are some things you can do to all of these things?

Wife: I don't get it...

Ryan: You can get them, right? You can pick them up. You can knock them over. You can burn them. You can apply those same exact verbs to any of the objects sitting there.

Wife: Okay... so?

Ryan: Well, that's important. What if instead of me being able to say to you, "get the cup," and "get the newspaper," and "get the remote"; what if instead we needed to come up with different verbs for each of the nouns? I couldn't use the word "get" universally, but instead had to think up a new word for each verb/noun combination.

Wife: Wow! That's weird.

Ryan: Yes, it is. Our brains are somehow smart enough to know that the same verbs can be applied to many different nouns. Some verbs are more specific than others and apply only to a small set of nouns. For instance, I can't drive a cup and I can't drink a car. But some verbs are almost universal like GET, PUT, and DELETE.

Wife: You can't DELETE a cup.

Ryan: Well, okay, but you can throw it away. That was another joke, right?

Wife: Yeah.

Ryan: So anyway, HTTP--this protocol Fielding and his friends created--is all about applying verbs to nouns. For instance, when you go to a web page, the browser does an HTTP GET on the URL you type in and back comes a web page.

Web pages usually have images, right? Those are separate resources. The web page just specifies the URLs to the images and the browser goes and does more HTTP GETs on them until all the resources are obtained and the web page is displayed. But the important thing here is that very different kinds of nouns can be treated the same. Whether the noun is an image, text, video, an mp3, a slideshow, whatever. I can GET all of those things the same way given a URL.

Wife: Sounds like GET is a pretty important verb.

Ryan: It is. Especially when you're using a web browser because browsers pretty much justGET stuff. They don't do a lot of other types of interaction with resources. This is a problem because it has led many people to assume that HTTP is just for GETing. But HTTP is actually ageneral purpose protocol for applying verbs to nouns.

Wife: Cool. But I still don't see how this changes anything. What kinds of nouns and verbs do you want?

Ryan: Well the nouns are there but not in the right format.

Think about when you're browsing around amazon.com looking for things to buy me for Christmas. Imagine each of the products as being nouns. Now, if they were available in a representation that a machine could understand, you could do a lot of neat things.

Wife: Why can't a machine understand a normal web page?

Ryan: Because web pages are designed to be understood by people. A machine doesn't care about layout and styling. Machines basically just need the data. Ideally, every URL would have a human readable and a machine readable representation. When a machine GETs the resource, it will ask for the machine readable one. When a browser GETs a resource for a human, it will ask for the human readable one.

Wife: So people would have to make machine formats for all their pages?

Ryan: If it were valuable.

Look, we've been talking about this with a lot of abstraction. How about we take a real example. You're a teacher - at school I bet you have a big computer system, or three or four computer systems more likely, that let you manage students: what classes they're in, what grades they're getting, emergency contacts, information about the books you teach out of, etc. If the systems are web-based, then there's probably a URL for each of the nouns involved here: student, teacher, class, book, room, etc. Right now, getting the URL through the browser gives you a web page. If there were a machine readable representation for each URL, then it would be trivial to latch new tools onto the system because all of that information would be consumable in a standard way. It would also make it quite a bit easier for each of the systems to talk to each other. Or, you could build a state or country-wide system that was able to talk to each of the individual school systems to collect testing scores. The possibilities are endless.

Each of the systems would get information from each other using a simple HTTP GET. If one system needs to add something to another system, it would use an HTTP POST. If a system wants to update something in another system, it uses an HTTP PUT. The only thing left to figure out is what the data should look like.

Wife: So this is what you and all the computer people are working on now? Deciding what the data should look like?

Ryan: Sadly, no. Instead, the large majority are busy writing layers of complex specifications for doing this stuff in a different way that isn't nearly as useful or eloquent. Nouns aren't universal and verbs aren't polymorphic. We're throwing out decades of real field usage and proven technique and starting over with something that looks a lot like other systems that have failed in the past. We're using HTTP but only because it helps us talk to our network and security people less. We're trading simplicity for flashy tools and wizards.

Wife: Why?

Ryan: I have no idea.

Wife: Why don't you say something?

Ryan: Maybe I will.

b) Código

b1) Código ejemplo de un método del servicio REST

```
// servicio bibliotecas, metodos relacionados con la informacion relacionada de bibliotecas

@Path("/biblioteca")
public class servicioBiblioteca {
/*Obtener todas las bibliotecas de un usuario determinado*/

        @GET
        @Produces({ MediaType.APPLICATION_JSON })
        public Response BuscarBibliotecasUsuario(@HeaderParam("Authorization") String jwtValidate) throws
ExpiredJwtException, UnsupportedJwtException, MalformedJwtException, IllegalArgumentException,
UnsupportedEncodingException, MalformedClaimException, JoseException, InvalidJwtException {

                System.out.println("buscar todas las bibliotecas ");
                JWT token = new JWT();
                if (token.validarToken(jwtValidate).getStatus()==200){
                        String[] infoToken = token.infoToken(jwtValidate);
                        String newToken = infoToken[0];
                        String nombreUsuario = infoToken[1];

                try {
                Bibliotecas numBibliotecas = new Bibliotecas();
                List<Bibliotecas> list = new ArrayList<Bibliotecas>();


      Connection c = null;
      String sql = "SELECT * FROM Bibliotecas WHERE idUsuario=(SELECT idUsuario FROM Usuario
WHERE nombre='"+nombreUsuario+"')";
      try {

         c = ConexionBD.GetConnection();
         Statement s = c.createStatement();
         ResultSet rs = s.executeQuery(sql);
         while (rs.next()) {
            list.add(processRow(rs));

         }

      } catch (SQLException e) {

         e.printStackTrace();
         throw new RuntimeException(e);
                } finally {
                        ConexionBD.close(c);
                }
```

```
        //numBibliotecas.setNumBibliotecas(list.size());
        //list.add(numBibliotecas);
        GenericEntity<List<Bibliotecas>> entity = new GenericEntity<List<Bibliotecas>>(list) {};

        return Response.ok(entity).header("Authorization", newToken).build();

                        } catch (Exception e) {
                    return Response.status(Response.Status.PRECONDITION_FAILED).build();

                }

            }
            else{
                    return Response.status(Response.Status.UNAUTHORIZED).entity("Error en el token").build();

            }
    }
}
```

b2) Código ejemplo de interacción del cliente Web con el servicio

GET:

```
<?php

session_start();

//$data = "usuario=".$usuario."&contrasenia=".$contrasenia;
$service_url = 'http://localhost:8080/gestionBibliotecas/bibliotecas/miBiblioteca/'.
$_SESSION["idBiblioteca"].'/libros';

$headers = array(
    'Content-Type: application/x-www-form-urlencoded',
    'Authorization: '.$_SESSION["token"],
);

$handle = curl_init($service_url);

curl_setopt($handle, CURLOPT_RETURNTRANSFER, true);
curl_setopt($handle, CURLOPT_HTTPHEADER, $headers);
//curl_setopt($handle, CURLOPT_HEADER ,1);
curl_setopt($handle, CURLOPT_HEADER, false);
curl_setopt($handle, CURLOPT_RETURNTRANSFER,1);

$curl_response = curl_exec($handle);
$httpcode = curl_getinfo($handle, CURLINFO_HTTP_CODE);

if ($curl_response === false) {
    $info = curl_getinfo($curl);
    curl_close($curl);
    die('error occured during curl exec. Additioanl info: ' . var_export($info));
}

curl_close($handle);

$decoded = json_decode($curl_response,true);
//var_dump($decoded);
//print_r($decoded);

?>
```

POST:

```
$data = "nombre=".$nombreBiblioteca."&descripcion=".$descripcion;
$service_url = 'http://localhost:8080/gestionBibliotecas/bibliotecas/biblioteca/'.
$_SESSION["idBiblioteca"];

$headers = array(
   'Content-Type: application/x-www-form-urlencoded',
   'Authorization: '.$_SESSION["token"],
);

$handle = curl_init($service_url);
curl_setopt($handle, CURLOPT_POST, true);
curl_setopt($handle, CURLOPT_POSTFIELDS, $data);
curl_setopt($handle, CURLOPT_HTTPHEADER, $headers);
curl_setopt($handle, CURLOPT_HEADER ,1);
curl_setopt($handle, CURLOPT_RETURNTRANSFER,1);

$curl_response = curl_exec($handle);

list($headers, $response) = explode("\r\n\r\n", $curl_response, 2);
// $headers now has a string of the HTTP headers
// $response is the body of the HTTP response

$headers = explode("\n", $headers);
foreach($headers as $header) {
   if (stripos($header, 'Authorization:') !== false) {
         $token = substr($header,14);

   }
}
$httpcode = curl_getinfo($handle, CURLINFO_HTTP_CODE);
curl_close($handle);

}
```

b3) Código token JWT

```
public Response generaToken (String nombre, String contrasenia) throws UnsupportedEncodingException{

                System.out.println("nombre: " + nombre);
                System.out.println("contrasenia " + contrasenia);

                //comprobacion si los parametros son null

                if (nombre == null) {
                    return Response.status(Response.Status.UNAUTHORIZED).entity("campo nombre
vacio").build();
                }

                if (contrasenia == null) {
                    return Response.status(Response.Status.UNAUTHORIZED).entity("campo contraseña
vacio").build();
                }

                UsuariosDAO usrDAO = new UsuariosDAO();
                Usuario user = usrDAO.UsuarioValido(nombre, contrasenia);
                // comprobamos que el usuario user es correcto o no. Si es null --> no es correcto

                if (user == null){
                        return Response.status(Response.Status.UNAUTHORIZED).entity("Usuario o contraseña
no son validos o no tiene permisos").build();
                }

                //Si todo ha ido bien, empezamos a generar el token
                //Header
                int IdUsuario=user.getIdUsuario();
                long nowMillis = System.currentTimeMillis();
            Date now = new Date(nowMillis);
            long expMillis = nowMillis + 6000*100;

                String jwt = Jwts.builder()
                                .setSubject("1234567890")
                                .setId("6c244e52-83e9-4069-aa03-422aa8b987bb")
                                .setExpiration(new Date(expMillis))
                                .claim("nombre", nombre)
                                .claim("contraseña", contrasenia)
                                .claim("id",IdUsuario)
                                .signWith(SignatureAlgorithm.HS256, "ggt23hwn24sk32".getBytes("UTF-8"))
                                .compact();

                return Response.status(Response.Status.CREATED).entity("Usuario autenticado
correctamente").header("Authorization", jwt).build();
  }
```

b4) Código tablas MySQL

```
CREATE TABLE `miBiblioteca`.`Usuario` (
  `idUsuario` INT NOT NULL AUTO_INCREMENT,
  `nombre` VARCHAR(100) NOT NULL,
  `contraseña` VARCHAR(45) NOT NULL,
  `email` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`idUsuario`),
  UNIQUE INDEX `idUsuario_UNIQUE` (`idUsuario` ASC));
```

```
CREATE TABLE `miBiblioteca`.`Bibliotecas` (
  `idBibliotecas` INT NOT NULL AUTO_INCREMENT,
  `idUsuario` INT,
  `nombre` VARCHAR(100) NOT NULL,
  `descripcion` VARCHAR(250) NULL,
  PRIMARY KEY (`idBibliotecas`),
  UNIQUE INDEX `idBibliotecas_UNIQUE` (`idBibliotecas` ASC),
  CONSTRAINT `usuario`
    FOREIGN KEY (`idUsuario`)
    REFERENCES `miBiblioteca`.`Usuarios` (`idUsuario`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION);
```

```
CREATE TABLE `miBiblioteca`.`Libros` (
  `idLibros` INT NOT NULL AUTO_INCREMENT,
  `idBiblioteca` INT NOT NULL,
  `titulo` VARCHAR(45) NOT NULL,
  `autor` VARCHAR(45) NOT NULL,
  `genero` VARCHAR(45) NOT NULL,
  `saga` VARCHAR(45) NULL,
  `editorial` VARCHAR(45) NOT NULL,
  `paginas` INT NULL,
  `ISBN` VARCHAR(45) NULL,
  `sinopsis` VARCHAR(250) NULL,
  `fila` INT NULL,
  `columna` INT NULL,
  `codGenerado` VARCHAR(45) NULL,
  `imagen` BLOB NULL,
  PRIMARY KEY (`idLibros`),
  UNIQUE INDEX `idLibros_UNIQUE` (`idLibros` ASC),
  UNIQUE INDEX `codGenerado_UNIQUE` (`codGenerado` ASC),
  CONSTRAINT `biblioteca`
    FOREIGN KEY (`idBiblioteca`)
    REFERENCES `miBiblioteca`.`Bibliotecas` (`idBibliotecas`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION);Escribe para introducir texto
```