



Promoción del desarrollo de SW libre en un entorno de
calidad y confianza adaptando las metodologías, procesos,
modelos de negocio y últimas tecnologías

FIT-350503-2007-7

D4.3.1
**Especificación de la plataforma de integración en la forja
Vulcano/EzForge**

Editor: Telefónica I + D
Editor: Universidad Rey Juan Carlos
Revisor: Telefónica I+D

Fecha límite del entregable: 29/02/2008

Fecha de entrega: 29/02/2008

Este trabajo se licencia bajo Creative Commons Attribution-Share Alike 3.0.

Para ver una copia de esta licencia, visite <http://creativecommons.org/licenses/by-sa/3.0/> o envíe una carta a Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

Este trabajo está parcialmente financiado por el Ministerio de Industria, Turismo y Comercio español.

Historial de Cambios

Versión	Fecha	Estado	Autor (Partner)	Descripción
0.1	16/11/2007	Borrador	Álvaro Polo (URJC)	Introducción, Contexto tecnológico
0.2	30/11/2007	Borrador	Álvaro Polo (URJC)	Especificación de la plataforma de integración (sin completar)
0.3	19/12/2007	Borrador	Irenka Redondo (TID)	Se completan los subapartados “Capa de integración de servicios” y “Núcleo de servicios”
0.4	20/12/2007	Borrador	Luis de la Fuente (TID)	Actualización de los recursos del subapartado “Capa de recursos”
1.0	29/02/2008	Final	Irenka Redondo (TID)	Revisión

RESUMEN EJECUTIVO

Este entregable describe de una manera más profunda la parte servidor de la arquitectura definida para el núcleo de la forja. Por un lado se define la capa de recursos que modelan las funcionalidades proporcionadas por la forja, y por otro, se describen las capas de servicios, que están formadas por el nivel de integración de servicios en la forja y por el núcleo de servicios para el funcionamiento interno de la forja.

Información del Documento

Proyecto FIT Número	FIT-350503-2007-7	Acrónimo	Vulcano
Título completo	Promoción del desarrollo de SW libre en un entorno de calidad y confianza adaptando las metodologías, procesos, modelos de negocio y últimas tecnologías		
URL	http://www.ines.org.es/vulcano		
URL del documento			

Entregable	Número	D4.3.1	Título	Especificación de la plataforma de integración en la forja de Vulcano/Ezforge
Paquete de Trabajo	Número	PT4	Título	Arquitectura E2.0 para la forja Vulcano
Tarea	Número	T4.3	Título	Integración en la parte servidor

Fecha de Entrega	Contractual	/02/2008	Entregado	/02/2008
Estado	Versión X, fecha dd/mm/yyyy		final	<input checked="" type="checkbox"/>
Tipo	Informe <input type="checkbox"/> Demo <input type="checkbox"/> Otro <input checked="" type="checkbox"/>			
Nivel de Diseminación	Público <input checked="" type="checkbox"/> Consorcio <input type="checkbox"/>			
Resumen (para diseminación)	Por un lado se define la capa de recursos que modelan las funcionalidades proporcionadas por la forja, y por otro, se describen las capas de servicios, que están formadas por el nivel de integración de servicios en la forja y por el núcleo de servicios para el funcionamiento interno de la forja.			
Palabras Clave	Recursos, integración, núcleo servicios			

Autores (Partner)	Álvaro Polo (URJC), Luis de la Fuente (TID), Irenka Redondo (TID)		
Responsable de Autoría	Irenka Redondo		Email iredondo@tid.es
	Partner	TID	Tfno +34 91 337 39 14

TABLA DE CONTENIDOS

Índice de contenido

RESUMEN EJECUTIVO.....	3
TABLA DE CONTENIDOS.....	5
1 INTRODUCCIÓN.....	7
2 CONTEXTO TECNOLÓGICO.....	8
2.1 Arquitecturas orientadas a servicios	9
2.2 Principio arquitectónico REST.....	11
3 ESPECIFICACIÓN DE LA PLATAFORMA.....	13
3.1 Capa de recursos.....	13
3.1.1 Categoría de Usuarios.....	13
3.1.1.1 Usuarios.....	13
3.1.1.2 Usuario.....	14
3.1.1.3 Habilidades.....	15
3.1.1.4 Habilidad.....	16
3.1.2 Categoría de Privilegios.....	16
3.1.2.1 Roles.....	17
3.1.2.2 Role.....	17
3.1.2.3 Permisos.....	18
3.1.2.4 Permiso.....	19
3.1.3 Categoría de Wiki.....	19
3.1.3.1 Wikis.....	20
3.1.3.2 Wiki.....	20
3.1.3.3 Artículos de Wiki.....	21
3.1.3.4 Artículo de Wiki.....	22
3.1.4 Categoría de Código Fuente.....	23
3.1.4.1 Navegador de Código.....	23
3.1.4.2 Fichero.....	24
3.1.4.3 Revisión.....	26
3.1.5 Categoría de Tareas.....	27
3.1.5.1 Tickets.....	27
3.1.5.2 Ticket.....	28
3.1.5.3 Hitos.....	28
3.1.5.4 Hito.....	29
3.1.5.5 Componentes.....	30
3.1.5.6 Componente.....	30
3.1.5.7 Versiones.....	31
3.1.5.8 Versión.....	32
3.1.6 Categoría de Proyectos.....	32
3.1.6.1 Proyectos.....	32
3.1.6.2 Proyecto.....	33

3.1.6.3 Usuarios de Proyecto.....	34
3.1.6.4 Proyectos de Usuario.....	34
3.1.6.5 Usuario de Proyecto.....	35
3.1.7 Categoría de Backends.....	36
3.1.7.1 Backends.....	36
3.1.7.2 Backend.....	37
3.1.7.3 Usuarios de Backend.....	37
3.1.7.4 Backends de Usuario.....	38
3.1.7.5 Usuario de Backend.....	39
3.2Capa de Integración de Servicios.....	40
3.2.1 Modelos de acceso a la información.....	41
3.2.2 Desarrollo de adaptadores.....	42
3.2.3 Integración de GForge.....	43
3.2.4 Integración de MediaWiki.....	43
3.2.5 Integración de servicios ad-hoc.....	44
3.3 Núcleo de servicios.....	44
3.3.1 Gestión de usuarios.....	44
3.3.2 Autenticación.....	44
3.3.3 Gestión de funcionalidades y herramientas.....	46
ANEXO : API SOAP DE GFORGE.....	47
REFERENCIAS	64

1 INTRODUCCIÓN

La popularización del movimiento del software libre a mediados de los años noventa trajo consigo nuevos modelos de desarrollo de software basados en principios de colaboración. Estas nuevas técnicas descentralizadas requerían de una infraestructura tecnológica capaz de dar soporte al desarrollo colaborativo en el contexto de una comunidad, con equipos de desarrollo dispersos a lo largo del globo. Para satisfacer estas necesidades, hacen su aparición en escena diversos productos software orientados a proporcionar las herramientas y servicios necesarios para garantizar un proceso de desarrollo eficiente. Tal es el caso de productos como CVS o GNU Mailman, cuyo uso se extendió significativamente en las comunidades de software libre. A finales de la década de los noventa, la empresa VA Linux publica el servicio SourceForge.net como un sistema integrado de apoyo al ciclo de vida de procesos de código abierto, el cual pronto se convertiría en un referente en el mundo del software libre. La experiencia de SourceForge marca el comienzo de los llamados Entornos de Desarrollo Colaborativos (CDE, *Collaborative Development Environments*) o Forjas, herramientas diseñadas para satisfacer todas las necesidades relacionadas con los procesos de desarrollo en el contexto del software libre.

En los últimos años, la industria del software ha dirigido su mirada con atención al fenómeno del software libre. No son pocas las empresas que ven fructíferas oportunidades de negocio en el uso y desarrollo de tecnologías libres. Tal es el caso de la Comunidad Morfeo, que engloba a empresas, centros de investigación y universidades en un consorcio que persigue aprovechar las sinergias resultantes del desarrollo de software empleando fórmulas de código abierto. En este nuevo contexto, en el cual la capitalización de los resultados juega un papel fundamental, aparecen nuevas necesidades y requisitos que deben ser satisfechos por las herramientas de desarrollo colaborativo.

Por este motivo, uno de los compromisos fundamentales del Proyecto Vulcano es ofrecer una plataforma que facilite la implantación de nuevos procesos de desarrollo basados en metodologías propias de entornos corporativos adaptadas al desarrollo colaborativo. Este objetivo inspira el desarrollo de EzForge, que pretende marcar el camino de lo que será la nueva generación de CDEs.

Este documento recoge la especificación de la plataforma de integración en la que se basa EzForge, la cual como veremos a través del mismo se encuentra fuertemente sustentada en tecnologías propias de la Web 2.0, tales como el empleo de clientes dinámicos mediante componentes *mashup* y servicios web ligeros basados en los principios arquitectónicos REST.

Hemos denominado EzForge al núcleo de forja que contendrá las funcionalidades básicas y habituales de una forja de desarrollo, y a partir del cual se podrán integrar nuevas herramientas que permitan aumentar la calidad de los proyectos desarrollados y formando todo en conjunto la forja Vulcano. Por esta razón, a lo largo del documento se puede hablar indistintamente de EzForge, Vulcano o núcleo de la forja.

2 CONTEXTO TECNOLÓGICO

Uno de los objetivos fundamentales de EzForge es la integración de componentes *legacy*, aquellos programas o sistemas de información cuya concepción no incluyera la posibilidad de ser integrados en sistemas ajenos. La integración de estos componentes proporcionaría un alto nivel de reutilización de código y de adaptabilidad y extensibilidad del sistema.

Por un lado, los modelos de datos de los componentes puede ser accedidos mediante un sistema integrado, lo que facilita el acceso a la información por parte de los usuarios. Este mismo objetivo es perseguido por los Entornos de Desarrollo Colaborativos (CDE) tales como GForge, el cual integra un sistema de información orientado a proyectos entre cuyas funcionalidades destacan el seguimiento de tareas y errores, publicación de productos software y documentación, foros de discusión, etc, las cuales se ven integradas con herramientas de Gestión de Código Fuente (SCM) tales como CVS o Subversion, listas de distribución de correo Mailman o un sistema integrado de gestión de usuarios con LDAP. Esta integración proporciona al usuario un punto de entrada único a la información a través de un espacio web, de tal manera que pueda llevarse a cabo tareas típicas de gestión de proyectos empleando una única herramienta. Para alcanzar este objetivo, G-Forge emplea un sistema de integración que le permite “atar” los distintos componentes que lo forman. Lamentablemente, dicha arquitectura es extremadamente inflexible al estar basada en *scripts* UNIX de tareas planificadas mediante el demonio *cron*. La escasa robustez de esta solución, unida a la baja calidad de la documentación hace prácticamente inviable añadir nuevos componentes a dicha arquitectura.

Por otro lado, las experiencias de la denominada Web 2.0 señalan un universo en el que la información es lo primero, por delante de los servicios proporcionados sobre la misma. En una Red donde lo fundamental es poder acceder a la información, el usuario deseará contar con sistemas capaces de proporcionar los mecanismos necesarios para poder navegar fácilmente por la misma, creando los enlaces y las ontologías necesarias para ello. Esta cuestión fundamental es también parte de la filosofía de integración de EzForge. Un ejemplo revelador es aquel en el cual un usuario participa en distintos proyectos de desarrollo de software los cuales se encuentran alojados en diversos CDEs. Ya que la información es lo fundamental, un modelo de integración idóneo será aquel que sea capaz de mostrar la información relativa a los proyectos de dicho usuario a través de un sistema único y cohesionado (fundamentalmente un sitio web), por ejemplo ofreciendo una lista de informes de error en la cual aparecen entradas tanto de un proyecto alojado en la Forja de Morfeo, como de otro hospedado en SourceForge.net, como de otro en la Forja de Rediris. La existencia de una aplicación web que permita acceder a la información de terceras partes al mismo tiempo que ofrece servicios fundamentales es uno de los factores principales que definen el modelo de integración de EzForge.

Estos objetivos descritos anteriormente, condicionan los requisitos del sistema de integración de EzForge, el cual deberá proporcionar una arquitectura que satisfaga estas necesidades. Como veremos a continuación, el estado del arte señala a las arquitecturas orientadas a servicios como uno de los principales modelos de integración que mejor se adaptan a este contexto tecnológico. A continuación, estudiaremos en detalle este tipo de arquitecturas, señalando aquellas características que las definen como grandes candidatas a formar la base de integración de EzForge. Posteriormente, introduciremos el concepto de arquitecturas REST como alternativa a los servicios web tradicionales debido a la innecesaria complejidad de estos últimos.

2.1 Arquitecturas orientadas a servicios

Las arquitecturas orientadas a servicios (SOA) ofrecen al desarrollador un marco de trabajo que le permite crear aplicaciones mediante una descomposición pobremente acoplada de sus elementos. Dichas arquitecturas dividen las aplicaciones en artefactos independientes denominados servicios, los cuales serán accedidos mediante interfaces basadas en protocolos y estándares abiertos, lo que garantiza la interoperabilidad entre los mismos independiente de plataforma y lenguaje de implementación. Esta interoperabilidad, unida al encapsulamiento derivado de la independencia de los servicios, se traduce en un bajo acoplamiento que garantiza una fácil evolución y adaptación del software.

Las arquitecturas orientadas a servicios emplean mecanismos RPC de paso de mensajes a través de la red, formando un sistema distribuido. Esta comunicación frecuentemente emplea protocolos estándar como HTTP para el paso de información entre los distintos servicios, lo cual permite que los servicios sean acoplados de manera completamente dinámica en tiempo de ejecución. Por otro lado, el uso de métodos estándar de representación de la información garantiza que los flujos de datos sean plenamente interoperables entre los distintos agentes que harán uso de los servicios. Estas dos características se traducen en:

- Independencia de plataforma. La interoperabilidad que ofrece el paso de mensajes y el uso de métodos de representación estándar permite que los servicios sean implementados con independencia de la plataforma. Esto elimina cualquier restricción respecto a hardware, sistema operativo o lenguaje de programación.
- Reutilización de componentes. El principio de descomposición permite al desarrollador reutilizar distintos servicios para sus aplicaciones. Incluso, en el caso concreto de servicios basados en el web, sería posible reutilizar servicios creados por terceras partes.
- Ciclos de vida independientes. Los servicios proporcionan interfaces conocidas de manera que los agentes que conforman el sistema puedan hacer uso de los mismos. Este hecho determina una clara separación respecto a la implementación, lo que permite que los servicios evolucionen independientemente unos de otros.

- Escalabilidad. La comunicación basada en paso de mensajes a través de la red facilita la distribución de los servicios a través de distintos nodos, lo que permite al desarrollador diseñar el sistema de tal manera que la carga de proceso se distribuya eficientemente.

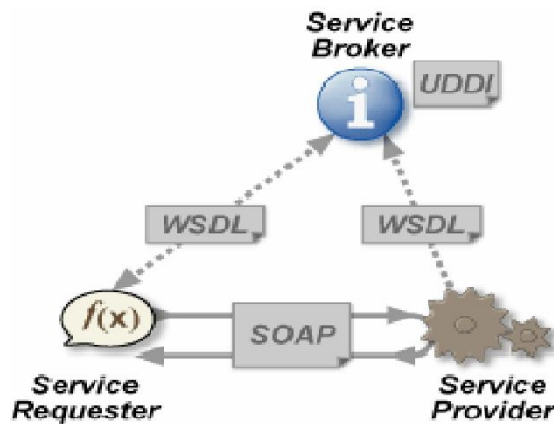
Estas características convierten a las arquitecturas basadas en servicios en soluciones especialmente idóneas para la construcción de sistemas que deban enfrentarse constantemente a modificaciones o adaptaciones gracias a la flexibilidad que ofrece su diseño. En el contexto de EzForge, al crearse una separación total entre interfaz e implementación, las arquitecturas basadas en servicios proporcionan un marco ideal para integrar componentes *legacy* en un sistema complejo.

Como hemos mencionado anteriormente, una de las piezas clave en las arquitecturas orientadas a servicios es la estandarización. Esto incluye tanto a los protocolos de comunicación mediante los cuales se implementa el paso de mensajes y la lógica RPC como a los métodos de representación de la información. Para acotar esta problemática, el World Wide Web Consortium (W3C) define una serie de normas y estándares orientados al desarrollo de Servicios Web.

El W3C define los servicios web como “sistemas software diseñados para dar soporte a la interoperabilidad entre máquinas a través de una red de computadores”. En la práctica, esta definición se emplea para designar a aquellas tecnologías que siguen un modelo arquitectónico basado en servicios empleando la Web como medio de comunicación. Estas tecnologías pueden ser clasificadas en función del papel que jueguen dentro del sistema distribuido en lo que se conoce como la Pila de Protocolos de Servicios Web:

- Protocolo de transporte. Es el responsable de dar soporte al paso de mensajes a través de la red de computadores. Muy frecuentemente se emplea HTTP y HTTPS para llevar a cabo esta tarea.
- Protocolo de mensaje. Es el responsable de representar la semántica de las operaciones invocadas sobre los servicios. Comúnmente se emplean protocolos estándar como XML-RPC o SOAP.
- Protocolo de descripción. Su tarea consiste en describir los distintos servicios disponibles para que puedan ser interpretados por las propias máquinas que hagan uso de los mismos. El estándar más empleado en la industria es el Web Service Description Language (WSDL).
- Protocolo de localización. La flexibilidad proporcionada por las arquitecturas SOA aumenta significativamente si los servicios pueden ser localizados dinámicamente por aquellos agentes que deseen hacer uso de los mismos. Esa es la tarea de este protocolo, cuyo mayor representante es UDDI (Universal Description, Discovery and Integration), estandarizado por OASIS.

Estos cuatro protocolos, ampliamente utilizados en la industria, forman los bloques constructivos básicos para la creación de aplicaciones distribuidas SOA.



2.2 Principio arquitectónico REST

Las arquitecturas orientadas a servicios web, pese a haber experimentado una rápida popularización en los últimos tiempos, cuentan con un importante número de detractores descontentos con algunos de sus principios básicos. Dejando al margen los aspectos relacionados con la falta de rendimiento propia de los protocolos de comunicación basados en XML, algunos expertos señalan la excesiva complejidad de los servicios web basados en RPC en general y de SOAP en particular.

En 2000 Roy T. Fielding acuñó el término REST (Representational State Transfer) por vez primera en su tesis doctoral. En su disertación, Fielding relacionaba el éxito de la Web con la propia sencillez de la misma, definiendo las pautas básicas del principio arquitectónico REST. De esta manera nace el concepto de Arquitectura Orientada a Recursos (ROA), que no deja de ser una particularización de las Arquitecturas Orientadas a Servicios (SOA), basado en los siguientes conceptos:

- Recursos. Un recurso es cualquier entidad con significado que pueda ser referenciada mediante una URI. En esta definición se engloban datos, documentos, imágenes, formularios, catálogos, operaciones, servicios, algoritmos, etc. En definitiva, cualquier elemento que resulte útil o relevante y tenga una URI asociada es considerado un recurso.
- Estado. Cada recurso tiene asociado un estado el cual puede ser consultado o modificado a través de su contenido. Típicamente, los métodos de representación estarán firmemente acotados y definidos, de manera que las entidades puedan interpretar fácilmente y sin trabas el estado de cada recurso. Para el intercambio de información, es habitual encontrar representaciones basadas en XML u otros formatos fácilmente interpretables por un navegador web, como HTML o JSON. Igualmente es habitual que el estado de un recurso referencia mediante una URI a otro recurso, creando así hipervínculos navegables por los agentes.

- Interfaces bien definidas. El principio arquitectónico REST establece cuatro métodos fundamentales de acceso y manipulación de la información, siendo la base de operacional de todos y cada uno de los recursos del sistema. Estas cuatro operaciones, análogas al popular cuarteto CRUD (Create, Read, Update y Delete) en los sistemas de bases de datos, son la de creación, lectura, modificación y eliminación del recurso, coincidiendo con los cuatro métodos del protocolo HTTP Get, Post, Put y Delete, respectivamente. Esta relación no es casual, ya que tanto REST como HTTP son productos fundamentados en el concepto de la Web.
- Protocolo de comunicación. REST establece la necesidad de emplear protocolos de comunicación que cumplan los siguientes requisitos:
 - Modelo cliente/servidor. Los roles de cliente (agente que solicita un servicio o información) y servidor (agente que los proporciona) están claramente definidos.
 - Sin estado. Cada operación llevada a cabo sobre un recurso es independiente de las demás, de tal manera que el protocolo no almacena información acerca de las mismas.
 - Con soporte de memoria cache. El estado de los recursos puede ser mantenido en memorias intermedias que permitan optimizar el acceso a la información.
- Arquitectura de capas. La posibilidad de disponer los recursos a través de capas permite al desarrollador definir arquitecturas en las cuales se haga una clara separación entre los distintos intermediarios que participen en el sistema, tales como proxies, adaptadores o cortafuegos.

Estos principios forman la base conceptual de REST. Fielding introdujo su idea de cómo deberían diseñarse los sistemas de información en la red, basándose en el concepto de transferencia de estado: los clientes acceden a las representaciones de los recursos, en las cuales se pueden encontrar hipervínculos a otros recursos, y estas representaciones irán cambiando el estado del cliente. De esta manera los clientes navegan en un universo de información llevando a cabo sus casos de uso.

En el contexto de los servicios web, las arquitecturas orientadas a recursos introducen una nueva forma de entender los sistemas de información. En contra de los sistemas basados en RPC como SOAP, existe la posibilidad de entender el universo de información como una red de recursos accesibles mediante interfaces bien definidas en lugar de acciones o comandos ejecutados a través de operaciones RPC. Esta nueva perspectiva simplifica notablemente el diseño de los sistemas, reduciendo la complejidad innecesaria de las tecnologías asociadas a los servicios web convencionales.

3 ESPECIFICACIÓN DE LA PLATAFORMA

Uno de los objetivos principales de EzForge es la integración de servicios de apoyo a las tareas de desarrollo de software. La forma en que estos servicios serán integrados es crítica a la hora de garantizar los niveles mínimos de flexibilidad y adaptabilidad que deseamos alcanzar. Por ello, es imprescindible ofrecer una arquitectura de integración que permita componer aplicaciones complejas que se adapten a las necesidades de los usuarios y a los procesos de desarrollo en que estos participen.

Cuando hablamos de entornos de desarrollo colaborativos, hablamos fundamentalmente de integración de servicios a través de las herramientas que componen los mismos. De esta forma, el desarrollador puede llevar a cabo las tareas definidas en el proceso de desarrollo empleando el conjunto de funcionalidades que le ofrece el CDE a través de dichas herramientas.

La forma en que estos servicios o herramientas se integran unos con otros es determinante para el desarrollador. Una buena forma de introducir el problema es comprender que pueden llegar a existir tantos procesos de desarrollo como proyectos existan. Cada organización o equipo puede escoger uno u otro en función de las características del proyecto, del presupuesto, los plazos de entrega, las necesidades del cliente o del propio equipo de desarrollador, etc. Cada uno de estos procesos de desarrollo puede hacer uso de distintas herramientas, de manera que las necesidades varían de uno a otro. Teniendo en cuenta dicha heterogeneidad entre distintos procesos, es importante que los CDEs ofrezcan la posibilidad de integrar aquellas herramientas que sean necesarias para el proceso escogido, a la vez que la incorporación de otras nuevas resulte sencilla y barata.

3.1 Capa de recursos

La capa de recursos de EzForge define los recursos necesarios para que los clientes puedan interoperar con el sistema a través del paso de mensajes HTTP.

Estos recursos ofrecen la abstracción de los servicios ofrecidos por EzForge. Por esta razón se han establecido una serie de categorías, atendiendo a funcionalidades que ofrecen, dentro de las cuales se han agrupado los recursos. Estas categorías se detallan a continuación.

3.1.1 Categoría de Usuarios

Esta categoría engloba recursos relacionados con la gestión de los usuarios registrados en la forja y las habilidades que éstos poseen. A continuación se detallan estos recursos.

3.1.1.1 Usuarios

Modela la lista de usuarios del sistema EzForge. Un recurso que modela una posible lista de usuarios de la forja sería algo así:

```
<?xml version="1.0" encoding="utf-8"?>
<users>
  <user id="id1">
    <name>Name of user 1</name>
    <uri xlink:type="simple" xlink:show="replace"
      xlink:href="http://domain.example.com/ezforge/vX.Y/users/id1">http://domain.example.com/ezforge/vX.Y/users/id1</uri>
  </user>
  <user id="id2">
    <name>Name of user 2</name>
    <uri xlink:type="simple" xlink:show="replace"
      xlink:href="http://domain.example.com/ezforge/vX.Y/users/id2">http://domain.example.com/ezforge/vX.Y/users/id2</uri>
  </user>
  <user id="id3">
    <name>Name of user 3</name>
    <uri xlink:type="simple" xlink:show="replace"
      xlink:href="http://domain.example.com/ezforge/vX.Y/users/id3">http://domain.example.com/ezforge/vX.Y/users/id3</uri>
  </user>
</users>
```

El recurso se encuentra disponible en uris del tipo `http://domain.example.com/ezforge/vX.Y/users/`, y las operaciones que se tienen implementadas para este recurso son las siguientes:

- GET: devuelve un xml como el indicado arriba y representa la lista de usuarios que están registrados en EzForge
- POST, PUT, DELETE: no implementadas para este recurso.

3.1.1.2 Usuario

Este recurso modela un usuario de la lista en concreto. Un ejemplo de recurso que modela el detalle de un usuario sería algo así:

```
<?xml version="1.0" encoding="utf-8"?>
<user id="144">
  <sessionname>jg</sessionname>
  <name>Juan</name>
  <secondname>Gomez</secondname>
  <email>jg@gmail.com</email>
  <skills id="s144">
    <uri xlink:type="simple" xlink:show="replace"
      xlink:href="http://domain.example.com/ezforge/vX.Y/users/144/skills">http://domain.example.com/ezforge/vX.Y/users/144/skills</uri>
```

```
</skills>
<projects id="p144">
  <uri xlink:type="simple" xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/users/144/projects/">http://domain.example.com/ezf
orge/vX.Y/users/144/projects/</uri>
</projects>
<backends id="b144">
  <uri xlink:type="simple" xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/users/144/backends/">http://domain.example.com/e
zforge/vX.Y/users/144/backends/</uri>
</backends>
</user>
```

El recurso se encuentra disponible uris del tipo `http://domain.example.com/ezforge/vX.Y/users/{user_id}/`, y las operaciones que se tienen implementadas para este recurso son las siguientes:

- GET: devuelve un xml como el indicado arriba y representa el detalle de un usuario registrado en EzForge
- POST: permite la creación de un usuario en EzForge
- PUT: permite la modificación de un usuario existente en EzForge
- DELETE: permite el borrado de un usuario existente en de EzForge

3.1.1.3 *Habilidades*

Este recurso modela las habilidades que posee un determinado usuario. Un ejemplo de recurso que modele las habilidades de un determinado usuario (identificado en EzForge por el id 144) sería algo así:

```
<skills>
  <skill id="sk144_1">
    <name>Skill 1</name>
    <uri xlink:type="simple" xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/users/144/skills/sk144_1/">http://domain.example.co
m/ezforge/vX.Y/users/144/skills/sk144_1/</Uri>
  </skill>
  <skill id="sk144_2">
    <name>Skill 2</Name>
    <uri xlink:type="simple" xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/users/144/skills/sk144_2/">http://domain.example.co
m/ezforge/vX.Y/users/144/skills/sk144_2/</Uri>
  </skill>
  <skill id="sk144_N">
    <nameE>Skill N</name>
    <uri xlink:type="simple" xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/users/144/skills/sk144_N/">http://domain.example.co
```

```
m/ezforge/vX.Y/users/144/skills/sk144_N</uri>
</skill>
</skills>
```

El recurso se encuentra disponible en uris del tipo `http://domain.example.com/ezforge/vX.Y/users/{user_id}/skills/`, y las operaciones que se tienen implementadas para este recurso son las siguientes:

- GET: devuelve un xml como el indicado arriba y representa las habilidades que posee un determinado usuario de EzForge
- POST, PUT, DELETE: no implementadas para este recurso

3.1.1.4 *Habilidad*

Este recurso modela una habilidad en concreto de la lista de habilidades que posee un determinado usuario. Un ejemplo de recurso que modele una de las habilidades de un determinado usuario (identificado en EzForge por el id 144) sería algo así:

```
<skill id="sk144_1">
  <name>name for this skill</name>
  <title>development</title>
  <initDate>01/02/2006</initDate>
  <endDate>01/08/2007</endDate>
  <keyWords>java oracle</keyWords>
</skill>
```

El recurso se encuentra disponible en uris del tipo `http://domain.example.com/ezforge/vX.Y/users/{user_id}/skills/{skill_id}`, y las operaciones que se tienen implementadas para este recurso son las siguientes:

- GET: devuelve un xml como el indicado arriba y representa una determinada habilidad que posee un determinado usuario de EzForge
- POST: permite la creación de una nueva habilidad para un determinado usuario de EzForge
- PUT: permite la modificación de una determinada habilidad de un determinado usuario de EzForge.
- DELETE: permite el borrado de una habilidad perteneciente a un determinado usuario de EzForge

3.1.2 **Categoría de Privilegios**

Esta categoría engloba recursos relacionados con la gestión de los roles existentes en la forja así como los permisos que estos roles conllevan. A continuación se detallan estos recursos:

3.1.2.1 Roles

Este recurso modela los roles que se tienen definidos en un determinado proyecto, y que por tanto serán los que se le puedan asignar a un usuario que se registre en ese proyecto. Un ejemplo de recurso que representa la lista de roles que se tienen definidos dentro del proyecto vulcano puede ser el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<roles>
  <role id="jun_dev">
    <name>Junior Developer</name>
    <uri xlink:type="simple" xlink:show="replace"
      xlink:href="http://domain.example.com/ezforge/vX.Y/projects/vulcano/roles/jun_dev/">http://domain
      .example.com/ezforge/vX.Y/projects/vulcano/roles/jun_dev/</uri>
  </role>
  <role id="sen_dev">
    <name>Senior Developer</name>
    <uri xlink:type="simple" xlink:show="replace"
      xlink:href="http://domain.example.com/ezforge/vX.Y/projects/vulcano/roles/sen_dev/">http://domai
      n.example.com/ezforge/vX.Y/projects/vulcano/roles/sen_dev/</uri>
  </role>
</roles>
```

El recurso se encuentra disponible en uris del tipo `http://domain.example.com/ezforge/vX.Y/projects/{project_id}/roles/`, y las operaciones que se tienen implementadas para este recurso son las siguientes:

- GET: devuelve un xml como el indicado arriba y representa los roles que se tienen definidos para un determinado proyecto de EzForge
- POST, PUT, DELETE: no implementadas para este recurso

3.1.2.2 Role

Este recurso modela un determinado rol dentro de la lista de roles que existen para un determinado proyecto de EzForge. Un ejemplo de recurso que modele un rol sería algo así:

```
<?xml version="1.0" encoding="utf-8"?>
<role id="jun_dev">
  <name>Junior Developer</Name>
  <permissions>
    <uri xlink:type="simple" xlink:show="replace"
      xlink:href="http://domain.example.com/ezforge/vX.Y/projects/vulcano/roles/jun_dev/permissions/">http://
      domain.example.com/ezforge/vX.Y/projects/vulcano/roles/jun_dev/permissions/</uri>
  </permissions>
```

```
</role>
```

El recurso se encuentra disponible en uris del tipo `http://domain.example.com/ezforge/vX.Y/projects/{project_id}/roles/{role_id}`, y las operaciones que se tienen implementadas para este recurso son las siguientes:

- GET: devuelve un xml como el indicado arriba y representa un determinado rol dentro de un proyecto de EzForge
- POST: permite la creación de un nuevo rol dentro de un determinado proyecto de EzForge
- PUT: permite la modificación de un rol determinado dentro de proyecto de EzForge
- DELETE: permite el borrado de un rol perteneciente a un determinado proyecto de EzForge

3.1.2.3 Permisos

Este recurso modela los permisos asociados a un determinado rol dentro de un determinado proyecto de EzForge. Un ejemplo de recurso que representa los permisos que tiene el rol de desarrollador junior (junior developer, `jun_dev`) dentro del proyecto vulcano puede ser el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<permissions>
  <permission id="forum_dev">
    <name>Forum for Developers</name>
    <uri xlink:type="simple" xlink:show="replace"
      xlink:href="http://domain.example.com/ezforge/vX.Y/projects/vulcano/roles/jun_dev/permissions/forum_dev/">http://domain.example.com/ezforge/vX.Y/projects/vulcano/roles/jun_dev/permissions/forum_dev/</uri>
  </permission>
  <permission id="docs">
    <name>Documents</name>
    <uri xlink:type="simple" xlink:show="replace"
      xlink:href="http://domain.example.com/ezforge/vX.Y/projects/vulcano/roles/jun_dev/permissions/docs/">http://domain.example.com/ezforge/vX.Y/projects/vulcano/roles/jun_dev/permissions/docs/</uri>
  </permission>
</permissions>
```

El recurso se encuentra disponible en uris del tipo `http://domain.example.com/ezforge/vX.Y/projects/{project_id}/roles/{role_id}/permissions`, y las operaciones que se tienen implementadas para este recurso son las siguientes:

- GET: devuelve un xml como el indicado arriba y representa los permisos que tiene asociados un determinado rol dentro de un determinado proyecto de EzForge
- POST, PUT, DELETE: no implementadas para este recurso

3.1.2.4 Permiso

Este recurso representa un determinado privilegio perteneciente a la lista de privilegios que tiene asociado un rol dentro de un determinado proyecto de EzForge. Un ejemplo de recurso que modele un permiso sería algo así:

```
<?xml version="1.0" encoding="utf-8"?>
<permission>
  <id>forum_dev</id>
  <name>Forum for developers</name>
  <value>write</value>
  <possible-values>
    <value>read</value>
    <value>write</value>
    <value>read-write</value>
  </possible-values>
</permission>
```

El recurso se encuentra disponible en uris del tipo `http://domain.example.com/ezforge/vX.Y/projects/{project_id}/roles/{role_id}/permissions/{permission_id}`, y las operaciones que se tienen implementadas para este recurso son las siguientes:

- GET: devuelve un xml como el indicado arriba y representa un determinado permiso asociado a un rol dentro de un proyecto de EzForge
- POST: permite la creación de un nuevo permiso asociado a un rol dentro de un determinado proyecto de EzForge
- PUT: permite la modificación de un permiso asociado a un rol dentro de un determinado proyecto de EzForge
- DELETE: permite el borrado de un permiso asociado a un rol perteneciente a un determinado proyecto de EzForge

3.1.3 Categoría de Wiki

Esta categoría engloba recursos relacionados con la gestión de las wikis asociadas a los proyectos albergados en la forja. A continuación se detallan estos recursos:

3.1.3.1 Wikis

Este recurso representa la lista de wikis asociada a un determinado proyecto. Un ejemplo de recurso que representa la lista de wikis para un determinado proyecto de EzForge puede ser el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<wikis>
  <wiki id="id_wiki_1">
    <name>wiki 1 name</name>
    <uri xlink:type="simple" xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/projects/prj_id/wikis/id_wiki_1">http://domain.exam
ple.com/ezforge/vX.Y/projects/prj_id/wikis/id_wiki_1/</uri>
  </wiki>
  <wiki id="id_wiki_2">
    <name>wiki 2 name</name>
    <uri xlink:type="simple" xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/projects/prj_id/wikis/id_wiki_2">http://domain.exam
ple.com/ezforge/vX.Y/projects/prj_id/wikis/id_wiki_2/</uri>
  </wiki>
  <wiki id="id_wiki_3">
    <name>wiki 3 name</name>
    <uri xlink:type="simple" xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/projects/prj_id/wikis/id_wiki_3">http://domain.exam
ple.com/ezforge/vX.Y/projects/prj_id/wikis/id_wiki_3/</uri>
  </wiki>
</wikis>
```

El recurso se encuentra disponible en uris del tipo `http://domain.example.com/ezforge/vX.Y/projects/{project_id}/wikis/`. Las operaciones que se tienen implementadas para este recurso son las siguientes:

- GET: devuelve un xml como el indicado arriba y representa la lista de wikis de un determinado proyecto de EzForge
- POST, PUT, DELETE: no implementadas para este recurso

3.1.3.2 Wiki

Este recurso representa una wiki asociada a un determinado proyecto, el xml que define este recurso depende de la implementación interna de la wiki. Un ejemplo de recurso que representa una wiki para un determinado proyecto de EzForge puede ser el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<article>
  <title>main_page</title>
  <content><![CDATA[
```

```
<h2>Introduction</h2>
<p>Página principal de la wiki alojada en pallas.ls.fi.upm.es</p>
<p><a xlink:type="simple" xlink:show="replace" xlink:href="
http://domain.example.com/ezforge/vX.Y/projects/ezforge/wikis/ezforge_wiki/categories/Pruebas">Categoría: Pruebas</a> <a xlink:type="simple" xlink:show="replace" xlink:href="
http://domain.example.com/ezforge/vX.Y/projects/ezforge/wikis/ezforge_wiki/categories/EzForge">Categoría: EzForge</a></p>
]]>
</content>
</article>
```

El recurso se encuentra disponible en uris del tipo `http://domain.example.com/ezforge/vX.Y/projects/{project_id}/wikis/{wiki_id}`. Las operaciones que se tienen implementadas para este recurso son las siguientes:

- GET: devuelve un xml como el indicado arriba y representa la wiki de un determinado proyecto de EzForge
- POST: permite la creación de una wiki para un determinado proyecto de EzForge
- PUT: permite la modificación de una wiki para un determinado proyecto de EzForge
- DELETE: permite el borrado de una wiki para un determinado proyecto de EzForge

3.1.3.3 Artículos de Wiki

Este recurso representa la lista de artículos alojados en una determinada wiki asociada a un determinado proyecto de EzForge. Un ejemplo de recurso que representa una lista de artículos asociados a una wiki para un determinado proyecto de EzForge puede ser el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<articles>
  <article id="EzForge">
    <name>EzForge</name>
    <uri xlink:type="simple" xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/projects/ezforge/wikis/ezforge_wiki/categories/EzForge/articles/EzForge/">EzForge</uri>
  </article>
  <article id="EzForge Minute of Audioconference Oct 11 2007">
    <name>EzForge Minute of Audioconference Oct 11 2007</name>
    <uri xlink:type="simple" xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/projects/ezforge/wikis/ezforge_wiki/categories/EzForge/articles/EzForge Minute of Audioconference Oct 11 2007/">EzForge Minute of Audioconference Oct 11 2007</uri>
```

```
</article>
<article id="Main page">
  <name>Main page</name>
  <uri xlink:type='simple' xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/projects/ezforge/wikis/ezforge_wiki/categories/EzFor
ge/articles/Main page/">Main page</uri>
</article>
</articles>
```

El recurso se encuentra disponible en uris del tipo `http://domain.example.com/ezforge/vX.Y/projects/{project_id}/wiki/{wiki_id}/articles/`. Las operaciones que se tienen implementadas para este recurso son las siguientes:

- GET: devuelve un xml como el indicado arriba y representa la lista de artículos correspondientes a una wiki de un determinado proyecto de EzForge
- POST, PUT, DELETE: no implementadas para este recurso

3.1.3.4 Artículo de Wiki

Este recurso representa un artículo de una wiki asociada a un determinado proyecto de EzForge. Un ejemplo de recurso que representa un artículo de una wiki para un determinado proyecto de EzForge puede ser el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<article>
  <title>EzForge</title>
  <content><![CDATA[
    <h2>Introduction</h2>
    <p>This section is under construction.</p>
    <h2>Meetings</h2>
    <ul>
      <li><a xlink:type="simple" xlink:show="replace" xlink:href="
http://domain.example.com/ezforge/vX.Y/projects/ezforge/wikis/ezforge_wiki/articles/EzForge Minute of
Audioconference Oct 11 2007">EzForge Minute of Audioconference Oct 11 2007</a></li>
      <li><a xlink:type="simple" xlink:show="replace" xlink:href="
http://domain.example.com/ezforge/vX.Y/projects/ezforge/wikis/ezforge_wiki/articles/EzForge Minute of
Audioconference Oct 29 2007">EzForge Minute of Audioconference Oct 29 2007</a></li>
    </ul>
    <p><a xlink:type="simple" xlink:show="replace" xlink:href="
http://domain.example.com/ezforge/vX.Y/projects/ezforge/wikis/ezforge_wiki/categories/EzForge">Categoría: EzForge</a></p>
  ]]>
</content>
</article>
```

El recurso se encuentra disponible en uris del tipo `http://domain.example.com/ezforge/vX.Y/projects/{project_id}/wikis/{wiki_id}/categories/{category_id}/articles/{article_id}` o `http://domain.example.com/ezforge/vX.Y/projects/{project_id}/wikis/{wiki_id}/articles/{article_id}`, esta última en el caso de que el artículo no pertenezca a ninguna categoría. Las operaciones que se tienen implementadas para este recurso son las siguientes:

- GET: devuelve un xml como el indicado arriba y representa un artículo en una wiki de un determinado proyecto de EzForge
- POST: permite la creación de un artículo en una wiki para un determinado proyecto de EzForge
- PUT: permite la modificación de un artículo en una wiki para un determinado proyecto de EzForge
- DELETE: permite el borrado de un artículo en una wiki para un determinado proyecto de EzForge

3.1.4 Categoría de Código Fuente

Esta categoría engloba recursos relacionados con la gestión de código fuente, es decir, ficheros, versiones etc. A continuación se detallan estos recursos:

3.1.4.1 Navegador de Código

Este recurso modela un directorio de ficheros fuente perteneciente a un determinado proyecto alojado en EzForge. Un ejemplo de este recurso puede ser el siguiente:

```
?xml version="1.0" encoding="utf-8"?>
<cms directory="some_path">
  <cmsFiles>
    <file>
      <name>filename1</name>
      <uri xlink:type='simple' xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/projects/prj_id/cms/some_path/filename1/">http://do
main.example.com/ezforge/vX.Y/projects/prj_id/cms/some_path/filename1/</uri>
    </file>
    <file>
      <name>filename2</name>
      <uri xlink:type='simple' xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/projects/prj_id/cms/some_path/filename2/">http://do
main.example.com/ezforge/vX.Y/projects/prj_id/cms/some_path/filename2/</uri>
    </file>
    <file>
      <name>filename3</name>
      <uri xlink:type='simple' xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/projects/prj_id/cms/some_path/filename3/">http://do
```

```
main.example.com/ezforge/vX.Y/projects/prj_id/cms/some_path/filename3/</uri>
</file>
</cmsFiles>
</cms>
```

Para atender a las diferentes funcionalidades que ofrece el recurso se plantean las siguientes uris:

- uri base para acceder al recurso (raíz del recurso):
`http://domain.example.com/ezforge/vX.Y/projects/{project_id}/cms/`
- uri para obtener la vista de un determinado directorio del recurso:
`http://domain.example.com/ezforge/vX.Y/projects/{project_id}/cms/path_t_o_directory`. El mismo resultado se obtiene indicándole a la uri, mediante el parámetro *mode* con valor *view* (valor por defecto de la uri si no se indica parámetro *mode*), que se desea visualizar el contenido del directorio en cuestión
`http://domain.example.com/ezforge/vX.Y/projects/{project_id}/cms/path_t_o_directory?mode=view`
- uri para descargar un directorio completo con todos sus archivos en un fichero comprimido en formato zip:
`http://domain.example.com/ezforge/vX.Y/projects/{project_id}/cms/path_t_o_directory?mode=download`
- uri para navegar por distintas ramas (versionado):
`http://domain.example.com/ezforge/vX.Y/projects/{project_id}/cms/tags/branch_name`

Las operaciones que se tienen implementadas para este recurso son las siguientes:

- GET: muestra o descarga el directorio seleccionado
- POST: crea el directorio indicado por la uri en cuestión
- PUT: modifica el directorio indicado por la uri en cuestión
- DELETE: elimina el directorio indicado por la uri en cuestión

3.1.4.2 Fichero

Este recurso modela un determinado fichero dentro del sistema de control de versiones de un determinado proyecto de EzForge. Un ejemplo de este recurso podría ser:

```
<?xml version="1.0" encoding="utf-8"?>
<cmsFile>
  <fileName>/some_path/example.java</fileName>
  <branch>Default branch</branch>
```

```
<revisions>
  <revision number="4">
    <uri xlink:type='simple' xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/projects/prj_id/cms/some_path/example.java/revisio
n/4/">http://domain.example.com/ezforge/vX.Y/projects/prj_id/cms/some_path/example.java/revision/4/</
uri>
  </revision>
  <revision number="3">
    <uri xlink:type='simple' xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/projects/prj_id/cms/some_path/example.java/revisio
n/4/">http://domain.example.com/ezforge/vX.Y/projects/prj_id/cms/some_path/example.java/revision/4/</
uri>
  </revision>
  <revision number="2">
    <uri xlink:type='simple' xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/projects/prj_id/cms/some_path/example.java/revisio
n/4/">http://domain.example.com/ezforge/vX.Y/projects/prj_id/cms/some_path/example.java/revision/4/</
uri>
  </revision>
  <revision number="1">
    <uri xlink:type='simple' xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/projects/prj_id/cms/some_path/example.java/revisio
n/4/">http://domain.example.com/ezforge/vX.Y/projects/prj_id/cms/some_path/example.java/revision/4/</
uri>
  </revision>
</revisions>
</cmsFile>
```

Para atender a las diferentes funcionalidades que ofrece el recurso se plantean las siguientes uris:

- uri para acceder al recurso (última versión del fichero):
`http://domain.example.com/ezforge/vX.Y/projects/{project_id}/cms/path_t
o_file`
- uri para ver (misma funcionalidad que la anterior uri) o descargar el
fichero referenciado por el recurso, las funcionalidades se diferencian
mediante el parámetro *mode*:
`http://domain.example.com/ezforge/vX.Y/projects/{project_id}/cms/path_t
o_file?mode=view|download`

Las operaciones que se tienen implementadas para este recurso son las siguientes:

- GET: permite descargar o visualizar un determinado fichero perteneciente a un proyecto de EzForge
- POST: permite la creación de un determinado fichero en un proyecto de EzForge

- PUT: permite la modificación de un determinado fichero perteneciente a un proyecto de EzForge
- DELETE: permite el borrado de un determinado fichero perteneciente a un proyecto de EzForge

3.1.4.3 Revisión

Este recurso representa la versión de un determinado fichero perteneciente a un proyecto de EzForge. Un ejemplo de recurso que represente una determinada versión de un fichero podría ser:

```
<?xml version="1.0" encoding="utf-8"?>
<cmsRevisionFile>
  <revision>2</revision>
  <fileName>/some_path/example.java</fileName>
  <branch>Default branch</branch>
  <fileLength unit="bytes">504</fileLength>
  <date>Sat, 07 Sep 2006 00:00:01 GMT</date>
  <author>JohnSmith</author>
</cmsRevisionFile>
```

Para atender a las diferentes funcionalidades que ofrece el recurso se plantean las siguientes uris:

- uri para ver o descargar una determinada versión:
http://domain.example.com/ezforge/vX.Y/projects/{project_id}/cms/path_to_file/revisions/{rev_number}?mode=view|download
- uri para ver los detalles de la versión:
http://domain.example.com/ezforge/vX.Y/projects/{project_id}/cms/path_to_file/revisions/{rev_number}/
- uri para comparar dos versiones:
http://domain.example.com/ezforge/vX.Y/projects/{project_id}/cms/path_to_file/revisions/{rev_number}/diff/{other_rev_number}

Las operaciones que se tienen implementadas para este recurso son las siguientes:

- GET: devuelve un xml como el indicado arriba y representa una determinada versión de un fichero
- POST: permite la creación de una nueva versión para un determinado fichero perteneciente a un proyecto de EzForge
- PUT: no procede la modificación de una versión
- DELETE: no procede el borrado de una versión

3.1.5 Categoría de Tareas

Esta categoría engloba recursos relacionados con la gestión y planificación de las tareas a realizar por el equipo de desarrollo. A continuación se detallan estos recursos:

3.1.5.1 Tickets

Este recurso representa una lista de tickets asociada a un proyecto de EzForge o a un hito dentro del mismo. Un ejemplo de recurso que representa la lista de tickets que se tienen definidos dentro del proyecto vulcano puede ser el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<tickets>
  <ticket id="id_ticket1">
    <name>Ticket 1 name</name>
    <uri xlink:type='simple' xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/projects/vulcano/tickets/id_ticket1/">http://domain.ex
ample.com/ezforge/vX.Y/projects/vulcano/tickets/id_ticket1/</uri>
  </ticket>
  <ticket id="id_ticket2">
    <name>Ticket 2 name</name>
    <uri xlink:type='simple' xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/projects/vulcano/tickets/id_ticket2/">http://domain.ex
ample.com/ezforge/vX.Y/projects/vulcano/tickets/id_ticket2/</uri>
  </ticket>
</tickets>
```

Puesto que los tickets se pueden listar tanto a nivel de proyecto como a nivel de hito definido dentro del proyecto, se pueden consultar en uris del tipo:

- Tickets por proyecto de EzForge:
http://domain.example.com/ezforge/vX.Y/projects/{project_id}/tickets/
- Tickets por hito definido en un proyecto de EzForge:
http://domain.example.com/ezforge/vX.Y/projects/{project_id}/milestones/{milestone_id}/tickets/

Las operaciones que se tienen implementadas para este recurso son las siguientes:

- GET: devuelve un xml como el indicado arriba y representa los tickets que se tienen definidos para un determinado proyecto de EzForge (o para un hito)
- POST, PUT, DELETE: no implementadas para este recurso

3.1.5.2 Ticket

Este recurso modela un determinado ticket dentro de la lista de tickets que existen para un determinado proyecto de EzForge o un hito del mismo. Un ejemplo de recurso que modele un ticket sería algo así:

```
<?xml version="1.0" encoding="utf-8"?>
<ticket id="id1">
  <name>Milestone 1 name</name>
  <description>Short milestone description</description>
  <assign_to>User Name</assign_to>
  <milestone>Milestone Name</milestone>
  <priority>critical</priority>
  <type>task</type>
  <component>Component Name</component>
  <version>1.0</version>
</ticket>
```

Puesto que los tickets se pueden listar tanto a nivel de proyecto como a nivel de hito definido dentro del proyecto, se pueden consultar en uris del tipo:

- Ticket de un proyecto de EzForge:
http://domain.example.com/ezforge/vX.Y/projects/{project_id}/tickets/{ticket_id}
- Ticket de un hito definido en un proyecto de EzForge:
http://domain.example.com/ezforge/vX.Y/projects/{project_id}/milestones/{mileston_id}/tickets/{ticket_id}

Las operaciones que se tienen implementadas para este recurso son las siguientes:

- GET: devuelve un xml como el indicado arriba y representa un determinado ticket de un proyecto de EzForge (o de un hito)
- POST: permite la creación de un nuevo ticket dentro de un proyecto de EzForge (o dentro de un hito)
- PUT: permite la modificación de un ticket dentro de un proyecto de EzForge (o dentro de un hito)
- DELETE: permite el borrado de un ticket dentro de un proyecto de EzForge (o dentro de un hito)

3.1.5.3 Hitos

Este recurso representa la lista de hitos asociada a un proyecto de EzForge. Un ejemplo de recurso que representa la lista de hitos que se tienen definidos dentro del proyecto vulcano puede ser el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<Milestones>
  <Milestone id="id1">
    <Name>Milestone 1 name</Name>
    <uri xlink:type='simple' xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/projects/vulcano/milestones/id1/">http://domain.example.com/ezforge/vX.Y/projects/vulcano/milestones/id1/</uri>
  </Milestone>
  <Milestone id="id2">
    <Name>Milestone 2 name</Name>
    <uri xlink:type='simple' xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/projects/vulcano/milestones/id2/">http://domain.example.com/ezforge/vX.Y/projects/vulcano/milestones/id2/</uri>
  </Milestone>
</Milestones>
```

El recurso se encuentra disponible en uris del tipo `http://domain.example.com/ezforge/vX.Y/projects/{project_id}/milestones/`, y las operaciones que se tienen implementadas para este recurso son las siguientes:

- GET: devuelve un xml como el indicado arriba y representa la lista de hitos asociados a un determinado proyecto de EzForge
- POST, PUT, DELETE: no implementadas para este recurso

3.1.5.4 Hito

Este recurso modela un determinado hito dentro de la lista de hitos asociados a un determinado proyecto de EzForge. Un ejemplo de recurso que modele un hito sería algo así:

```
<?xml version="1.0" encoding="utf-8"?>
<milestone id="id1">
  <name>Milestone 1 name</name>
  <description>Short milestone description</description>
  <tickets>
    <uri xlink:type="simple" xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/projects/prj_id/milestones/id1/tickets/">http://domain.example.com/ezforge/vX.Y/projects/prj_id/milestones/id1/tickets/</uri>
  </tickets>
</milestone>
```

El recurso se encuentra disponible en uris del tipo `http://domain.example.com/ezforge/vX.Y/projects/{project_id}/milestones/{milestone_id}/`, y las operaciones que se tienen implementadas para este recurso son las siguientes:

- GET: devuelve un xml como el indicado arriba y representa un determinado hito perteneciente a un proyecto de EzForge
- POST: permite la creación de un nuevo hito asociado a un proyecto de EzForge
- PUT: permite la modificación de un hito perteneciente a un proyecto de EzForge
- DELETE: permite el borrado de un hito perteneciente a un proyecto de EzForge

3.1.5.5 Componentes

Este recurso representa la lista de componentes asociada a un proyecto de EzForge. Un ejemplo de recurso que representa la lista de componentes que se tienen definidos dentro del proyecto vulcano puede ser el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<components>
  <component id="id1">
    <name>Component 1 name</name>
    <uri xlink:type='simple' xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/projects/vulcano/components/id1/">http://domain.ex
ample.com/ezforge/vX.Y/projects/vulcano/components/id1/</uri>
  </component>
  <component id="id2">
    <name>Component 2 name</name>
    <uri xlink:type='simple' xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/projects/vulcano/components/id2/">http://domain.ex
ample.com/ezforge/vX.Y/projects/vulcano/components/id2/</uri>
  </component>
</components>
```

El recurso se encuentra disponible en uris del tipo `http://domain.example.com/ezforge/vX.Y/projects/{project_id}/components/`, y las operaciones que se tienen implementadas para este recurso son las siguientes:

- GET: devuelve un xml como el indicado arriba y representa la lista de componentes asociados a un determinado proyecto de EzForge
- POST, PUT, DELETE: no implementadas para este recurso

3.1.5.6 Componente

Este recurso modela un determinado componente dentro de la lista de componentes asociados a un determinado proyecto de EzForge. Un ejemplo de recurso que modele un componente sería algo así:

```
<component id="id1">
  <name>Component 1 name</name>
  <owner>Name of the owner of the component</owner>
  <description>Short component description</description>
</component>
```

El recurso se encuentra disponible en uris del tipo `http://domain.example.com/ezforge/vX.Y/projects/{project_id}/components/{component_id}/`, y las operaciones que se tienen implementadas para este recurso son las siguientes:

- GET: devuelve un xml como el indicado arriba y representa un determinado componente perteneciente a un proyecto de EzForge
- POST: permite la creación de un nuevo componente asociado a un proyecto de EzForge
- PUT: permite la modificación de un componente perteneciente a un proyecto de EzForge
- DELETE: permite el borrado de un componente perteneciente a un proyecto de EzForge

3.1.5.7 Versiones

Este recurso representa la lista de versiones correspondientes a un determinado componente asociado a un proyecto de EzForge. Un ejemplo de recurso que representa la lista de versiones correspondientes a un determinado componente dentro de un proyecto de EzForge puede ser el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<versions>
  <version id="id1">
    <name>version 1 name</name>
    <uri xlink:type='simple' xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/projects/prj_id/components/component_id/versions/id1/">http://domain.example.com/ezforge/vX.Y/projects/prj_id/components/component_id/versions/id1/</uri>
  </version>
  <version id="id2">
    <name>version 2 name</name>
    <uri xlink:type='simple' xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/projects/prj_id/components/component_id/versions/id2/">http://domain.example.com/ezforge/vX.Y/projects/prj_id/components/component_id/versions/id2/</uri>
  </version>
</versions>
```

El recurso se encuentra disponible en uris del tipo `http://domain.example.com/ezforge/vX.Y/projects/{project_id}/components/{component_id}/versions/`, y las operaciones que se tienen implementadas para este recurso son las siguientes:

- GET: devuelve un xml como el indicado arriba y representa la lista de versiones correspondientes a un componente asociado a un determinado proyecto de EzForge
- POST, PUT, DELETE: no implementadas para este recurso

3.1.5.8 Versión

Este recurso representa una determinada versión de un componente asociado a un determinado proyecto de EzForge. Un ejemplo de recurso que modele una versión sería algo así:

```
<?xml version="1.0" encoding="utf-8"?>
<version id="id1">
  <version>1.0.2</version>
  <description>Version description</description>
</version>
```

El recurso se encuentra disponible en uris del tipo `http://domain.example.com/ezforge/vX.Y/projects/{project_id}/components/{component_id}/versions/{version_id}/`, y las operaciones que se tienen implementadas para este recurso son las siguientes:

- GET: devuelve un xml como el indicado arriba y representa una determinada versión de un componente perteneciente a un proyecto de EzForge
- POST: permite la creación de un nueva versión de un componente asociado a un proyecto de EzForge
- PUT: permite la modificación de una versión de un componente perteneciente a un proyecto de EzForge
- DELETE: permite el borrado de una versión de un componente perteneciente a un proyecto de EzForge

3.1.6 Categoría de Proyectos

Esta categoría engloba recursos relacionados con la gestión de los proyectos almacenados en la forja. A continuación se detallan estos recursos:

3.1.6.1 Proyectos

Este recurso representa la lista de proyectos registrados en EzForge. Un ejemplo de recurso que represente la lista de proyectos registrados en EzForge puede ser el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<projects>
  <project id="id1">
    <name>Name of project 1</name>
    <uri xlink:type='simple' xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/projects/id1/">http://domain.example.com/ezforge/v
X.Y/projects/id1/</uri>
  </project>
  <project id="id2">
    <name>Name of project 2</name>
    <uri xlink:type='simple' xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/projects/id2/">http://domain.example.com/ezforge/v
X.Y/projects/id2/</uri>
  </project>
</projects>
```

El recurso se encuentra disponible en uris del tipo `http://domain.example.com/projects/`, y las operaciones que se tienen implementadas para este recurso son las siguientes:

- GET: devuelve un xml como el indicado arriba y representa la lista de proyectos registrados en EzForge
- POST, PUT, DELETE: no implementadas para este recurso

3.1.6.2 Proyecto

Este recurso representa un determinado proyecto de EzForge. Un ejemplo de recurso que modele un proyecto sería algo así:

```
<project id="1">
  <name>Name of project</name>
  <alias>Alias of project</alias>
  <description>A project short description</description>
  <scm>svn.domain.org</scm>
  <users>
    <uri xlink:type="simple" xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/projects/1/users/">http://domain.example.com/ezfor
ge/vX.Y/projects/1/users/</uri>
  </users>
</project>
```

El recurso se encuentra disponible en uris del tipo `http://domain.example.com/projects/{project_id}/`, y las operaciones que se tienen implementadas para este recurso son las siguientes:

- GET: devuelve un xml como el indicado arriba y representa un determinado proyecto de EzForge
- POST: permite la creación de un nuevo proyecto de EzForge
- PUT: permite la modificación de un proyecto de EzForge
- DELETE: permite el borrado de proyecto de EzForge

3.1.6.3 Usuarios de Proyecto

Este recurso representa la lista de usuarios que están registrados en un determinado proyecto de EzForge. Un ejemplo de recurso que represente la lista de usuarios de un proyecto de EzForge puede ser el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<project-users>
  <project-user id="idUser1">
    <name>Name of user 1</name>
    <uri xlink:type='simple' xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/projects/prj_id/users/idUser1/">http://domain.examp
e.com/ezforge/vX.Y/projects/prj_id/users/idUser1/</uri>
  </project-user>
  <project-user id="idUser2">
    <name>Name of user 2</name>
    <uri xlink:type='simple' xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/projects/prj_id/users/idUser2/">http://domain.examp
e.com/ezforge/vX.Y/projects/prj_id/users/idUser2/</uri>
  </project-user>
</project-users>
```

El recurso se encuentra disponible en uris del tipo http://domain.example.com/ezforge/vX.Y/projects/{project_id}/users/, y las operaciones que se tienen implementadas para este recurso son las siguientes:

- GET: devuelve un xml como el indicado arriba y representa la lista de usuarios registrados en un determinado proyecto de EzForge
- POST, PUT, DELETE: no implementadas para este recurso

3.1.6.4 Proyectos de Usuario

Este recurso representa la lista de proyectos en los que se encuentra registrado un determinado usuario de EzForge. Un ejemplo de recurso que represente la lista de proyectos en los que está registrado un usuario de EzForge puede ser el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<user-projects>
  <user-project id="idPrj1">
    <name>Name of project 1</name>
    <uri xlink:type='simple' xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/users/user_id/projects/idPrj1/">http://domain.
example.com/ezforge/vX.Y/users/user_id/projects/idPrj1/</uri>
  </user-project>
  <user-project id="idPrj2">
    <name>Name of project 2</name>
    <uri xlink:type='simple' xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/users/user_id/projects/idPrj2/">http://domain.
example.com/ezforge/vX.Y/users/user_id/projects/idPrj2/</uri>
  </user-project>
</user-projects>
```

El recurso se encuentra disponible en uris del tipo `http://domain.example.com/ezforge/vX.Y/users/{user_id}/projects/`, y las operaciones que se tienen implementadas para este recurso son las siguientes:

- GET: devuelve un xml como el indicado arriba y representa la lista de proyectos en los que está registrado un determinado usuario de EzForge
- POST, PUT, DELETE: no implementadas para este recurso

3.1.6.5 *Usuario de Proyecto*

Este recurso representa un usuario que pertenece a un determinado proyecto de EzForge, la información que aquí se muestra es información concreta referente a ese usuario en ese proyecto, y es distinta de la información genérica del recurso usuario de EzForge. Un ejemplo de recurso que represente un usuario dentro de un determinado proyecto puede ser el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<project-user>
  <user id="idUser">
    <uri xlink:type="simple" xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/users/idUser/">http://domain.example.com/ezforge/v
X.Y/users/idUser/</uri>
  </user>
  <project id="idProject">
    <uri xlink:type="simple" xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/projects/idProject/">http://domain.example.com/ezfo
rge/vX.Y/projects/idProject/</uri>
  </project>
  <role id="idRole">
    <uri xlink:type="simple" xlink:show="replace"
```

```
xlink:href="http://domain.example.com/ezforge/vX.Y/projects/idProject/roles/idRole/">http://domain.example.com/ezforge/vX.Y/projects/idProject/roles/idRole/</uri>  
</role>  
</project-user>
```

El recurso se encuentra disponible en uris del tipo `http://domain.example.com/ezforge/vX.Y/projects/{project_id}/users/{user_id}/` o la equivalente, `http://domain.example.com/ezforge/vX.Y/users/{user_id}/projects/{project_id}/`. Las operaciones que se tienen implementadas para este recurso son las siguientes:

- GET: devuelve un xml como el indicado arriba y representa la relación entre un determinado usuario y un determinado proyecto de Ezforge
- POST: permite la creación de la relación entre un determinado usuario y un determinado proyecto de Ezforge
- PUT: permite la modificación de la relación entre un determinado usuario y un determinado proyecto de Ezforge
- DELETE: permite el borrado de la relación entre un determinado usuario y un determinado proyecto de Ezforge

3.1.7 Categoría de Backends

Esta categoría engloba recursos relacionados con la gestión de los sistemas de backend integrados en EzForge. A continuación se detallan estos recursos:

3.1.7.1 Backends

Este recurso representa la lista de sistemas de backend soportados por EzForge. Un ejemplo de recurso que represente la lista de sistemas de backend soportados por EzForge puede ser el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>  
<backends>  
  <backend id="gforge_morfeo">  
    <name>Gforge MORFEO</name>  
    <uri xlink:type='simple' xlink:show="replace"  
xlink:href="http://domain.example.com/ezforge/backends/gforge_morfeo/">http://domain.example.com/ezforge/vX.Y/backends/gforge_morfeo/</uri>  
  </backend>  
  <backend id="trac_vulcano"/>  
    <name>Trac Vulcano</name>  
    <uri xlink:type='simple' xlink:show="replace"  
xlink:href="http://domain.example.com/ezforge/backends/trac_vulcano/">http://domain.example.com/ezforge/vX.Y/backends/trac_vulcano/</uri>  
  </backend>  
</backends>
```

El recurso se encuentra disponible en uris del tipo `http://domain.example.com/ezforge/vX.Y/backends/`, y las operaciones que se tienen implementadas para este recurso son las siguientes:

- GET: devuelve un xml como el indicado arriba y representa la lista de sistemas de backend soportados por EzForge
- POST, PUT, DELETE: no implementadas para este recurso

3.1.7.2 *Backend*

Este recurso representa un sistema de backend en concreto de los de la lista de sistemas de backend soportados por EzForge. Un ejemplo de recurso que represente un sistema de backend soportado por EzForge puede ser el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<backend>
  <id>gforge_morfeo</id>
  <description>Forge for the MORFEO community, based in GForge</description>
  <url>http://forge.morfeo-project.org</url>
  <uri-adaptor>http://host:port/gforge/v0.1</uri-adaptor>
  <users>
    <uri xlink:type="simple" xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/backends/gforge_morfeo/users/">http://domain.example.com/ezforge/vX.Y/backends/gforge_morfeo/users/</uri>
  </users>
</backend>
```

El recurso se encuentra disponible en uris del tipo `http://domain.example.com/ezforge/vX.Y/backends/{backend_id}/`, y las operaciones que se tienen implementadas para este recurso son las siguientes:

- GET: devuelve un xml como el indicado arriba y representa un sistema de backend soportado por EzForge
- POST: permite la creación de un sistema de backend soportado por EzForge
- PUT: permite la modificación de un sistema de backend soportado por EzForge
- DELETE: permite el borrado de un sistema de backend soportado por EzForge

3.1.7.3 *Usuarios de Backend*

Este recurso representa la lista de usuarios que pueden ver los proyectos almacenados en un determinado sistema de backend. Un ejemplo de recurso que represente la lista de sistemas de backend soportados por EzForge puede ser el siguiente:

```
<backend-users>
  <backend-user id="rlopez">
    <name>Roberto Lopez</name>
    <uri xlink:type='simple' xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/backends/gforge_morfeo/users/rlopez/">http://domai
n.example.com/ezforge/vX.Y/backends/gforge_morfeo/users/rlopez/</uri>
  </backend-user>
  <backend-user id="mgarcia">
    <name>Manuel Garcia</name>
    <uri xlink:type='simple' xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/backends/gforge_morfeo/users/mgarcia/">http://dom
ain.example.com/ezforge/vX.Y/backends/gforge_morfeo/users/mgarcia/</uri>
  </backend-user>
</backend-users>
```

El recurso se encuentra disponible en uris del tipo `http://domain.example.com/ezforge/vX.Y/backends/{backend_id}/users/`, y las operaciones que se tienen implementadas para este recurso son las siguientes:

- GET: devuelve un xml como el indicado arriba y representa la lista de usuarios registrados en un determinado sistema de backend
- POST, PUT, DELETE: no implementadas para este recurso

3.1.7.4 *Backends de Usuario*

Este recurso representa la lista de sistemas de backend en los que un usuario está registrado y por tanto a los cuales tendrá acceso. El esquema que rige este recurso es el siguiente:

Siguiendo este esquema un recurso que represente la lista de sistemas de backend que un usuario puede utilizar puede ser el siguiente:

```
<user-backends>
  <user-backend id="gforge_morfeo">
    <name>Gforge MORFEO</name>
    <uri xlink:type='simple' xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/users/rlopez/backends/gforge_morfeo/">http://domai
n.example.com/ezforge/vX.Y/users/rlopez/backends/gforge_morfeo/</uri>
  </user-backend>
  <user-backend id="trac_vulcano">
    <name>Trac Vulcano</name>
    <uri xlink:type='simple' xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/users/rlopez/backends/trac_vulcano/">http://domain.
example.com/ezforge/vX.Y/users/rlopez/backends/trac_vulcano/</uri>
  </user-backend>
</user-backends>
```

El recurso se encuentra disponible en uris del tipo `http://domain.example.com/ezforge/vX.Y/users/{user_id}/backends/`, y las operaciones que se tienen implementadas para este recurso son las siguientes:

- GET: devuelve un xml como el indicado arriba y representa la lista de sistemas de backend con los cuales un determinado usuario puede trabajar
- POST, PUT, DELETE: no implementadas para este recurso

3.1.7.5 Usuario de Backend

Este recurso representa un usuario de EzForge que pertenece a un determinado sistema de backend. Un ejemplo de recurso que represente un usuario dentro de un determinado sistema de backend puede ser el siguiente:

```
<backend-user>
  <backend id="gforge_morfeo">
    <uri xlink:type="simple" xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/backends/gforge_morfeo/">http://domain.example.c
om/ezforge/vX.Y/backends/gforge_morfeo/</uri>
  </backend>
  <user id="rlopez">
    <uri xlink:type="simple" xlink:show="replace"
xlink:href="http://domain.example.com/ezforge/vX.Y/users/rlopez/">http://domain.example.com/ezforge/v
X.Y/users/rlopez/</uri>
  </user>
  <username-backend>roberto_lopez</username-backend>
  <userpwd-backend>private</userpwd-backend>
</backend-user>
```

El recurso se encuentra disponible en uris del tipo `http://domain.example.com/ezforge/vX.Y/backends/{backend_id}/users/{user_id }` o la equivalente, `http://domain.example.com/ezforge/vX.Y/users/{user_id}/backends/{backend_id }`

Las operaciones que se tienen implementadas para este recurso son las siguientes:

- GET: devuelve un xml como el indicado arriba y representa la relación entre un determinado usuario y un determinado sistema de backend de Ezforge
- POST: permite la creación de la relación entre un determinado usuario y un determinado sistema de backend de Ezforge
- PUT: permite la modificación de la relación entre un determinado usuario y un determinado sistema de backend de Ezforge

- DELETE: permite el borrado de la relación entre un determinado usuario y un determinado sistema de backend de Ezforge

3.2 Capa de Integración de Servicios

La capa de integración de servicios es la capa responsable de adaptar los contenidos y métodos de acceso a la información proporcionados por las herramientas de forma que se ofrezca una API REST disponible para ser accedida desde EzForge. Como hemos visto anteriormente, identificamos con el término *back-end* a aquellos artefactos situados en la capa inferior de EzForge responsables de implementar las funcionalidades que satisfacen los casos de uso del sistema. Estos artefactos, según su naturaleza, pueden ser clasificados en:

- Servicios cerrados: componentes o fragmentos de software cuya concepción no consideraba el acceso a su información desde agentes externos. Su modelo de datos típicamente se encuentra almacenado en una base de datos o en ficheros con formato binario cuya especificación no es pública.
- Servicios abiertos: componentes o fragmentos de software cuyo diseño permite acceder a su modelo de datos o a su funcionalidad a través de una API bien definida.
 - Servicios basados en librerías. Permiten el acceso a la información e incluso a la funcionalidad a través de librerías diseñadas para dicho efecto.
 - Servicios con API basada en HTTP. Permiten el acceso a la información a través de protocolos de red basados en paso de mensajes a través HTTP siguiendo un modelo cliente/servidor.
 - Servicios con API SOAP/XML-RPC. Emplean los protocolos SOAP o XML-RPC para acceder a la información.
 - Servicios con API *RESTful*. Hacen uso de arquitecturas orientadas a recursos para acceder a la información.

Adicionalmente, podemos clasificar los servicios en función de las posibilidades que aportan a la hora de acceder a su información.

- Servicios de acceso libre. El acceso a su información está garantizado de manera abierta para todo agente que desee consultarla. Típicamente siguen un modelo de servicio abierto basado en APIs orientadas a servicios web. La implicación directa de este tipo de servicios es que ofrece la posibilidad de acceder e integrar la información sin contar con el permiso explícito de sus mantenedores, de manera que no añade restricción alguna al agente integrador. Un caso práctico de este modelo sería el poder obtener información de proyectos de desarrollo de un servicio GForge de manera que cada usuario pueda monitorizar sus proyectos en dicho sitio desde su interfaz de EzForge.
- Servicios de acceso restringido. El acceso a su información se encuentra restringido. Típicamente siguen un modelo cerrado de acceso a la

información, aunque pueden tratarse de modelos abiertos cuyo acceso se encuentre limitado bajo un sistema de autenticación. Su implicación inmediata es que precisan del consentimiento explícito de sus mantenedores para acceder a la información. Este consentimiento a menudo requiere que el agente integrador garantice ciertas restricciones, como por ejemplo ejecutarse en la misma máquina física que el servicio a integrar para garantizar la seguridad.

Estas clasificaciones son de gran utilidad a la hora de diseñar los modelos de adaptación y acceso a la información de los servicios.

3.2.1 Modelos de acceso a la información

En función del tipo de servicio que deseemos integrar, existen distintas soluciones para obtener el acceso a la información. Estas soluciones dependen de los conectores proporcionados por el servicio en cuestión para acceder a su modelo de datos o a su lógica de negocio.

- Conectores sobre TCP/IP. Estos conectores permiten acceder a la información a través de canales sobre una red telemática.
 - Conectores de SGBD SQL. Este tipo de conector permite acceder a la información a través de protocolos binarios sobre IP ofertados por un Sistema Gestor de Bases de Datos. De esta forma, los agentes integradores pueden efectuar consultas SQL sobre la base de datos relacional que contiene la información del servicio integrado. Para garantizar la seguridad de los datos, normalmente este tipo de servicios son restringidos, limitando el acceso a estos conectores a sus propios componentes lógicos. Esto suele impedir el acceso a la información a cualquier agente integrador, siendo necesaria la intervención de los mantenedores para autorizar dicho acceso.
 - Conectores SOAP/XML-RPC. Este tipo de conector permite acceder a la información en forma de servicios web a través de los protocolos SOAP o XML-RPC. Normalmente permiten un acceso abierto y libre a la información sin restricciones para el agente integrador.
 - Conectores RESTful. Este tipo de conector ofrece acceso a la información en forma de recursos web semánticos tales como canales RDF o APIs REST. Se trata de servicios abiertos y libres que no añaden restricciones a los agentes integradores.
 - Conectores HTTP/HTML. Este tipo de conector proporciona una vista renderizada en formato HTML de la información sobre HTTP. La función de estos conectores es la de interactuar con agentes humanos a través de contenidos web, motivo por el cual la información no es fácilmente interpretable por agentes basados en cómputo. Al tratarse de información normalmente accesible por canales abiertos, cualquier agente integrador podría acceder a la información sin apenas restricciones más allá de la propia autenticación del usuario en el sistema web. Sin embargo, la interpretación de los datos renderizados en HTML puede ser

extremadamente compleja y costosa, además de estar sujeta a los posibles cambios de estilo de los mantenedores del servicio.

- Conectores sobre sistemas de ficheros. El acceso a la información viene proporcionado por un sistema de ficheros que contiene los datos que se desea integrar. Suelen ser proporcionados por servicios cerrados y restringidos cuyo modelo de datos se almacena en ficheros. Por tanto, el acceso a la información está sujeto a la autorización de los mantenedores del servicio y a las limitaciones técnicas del sistema.

Estos modelos de acceso a la información determinarán la forma en que los agentes integradores accedan a la información de los *back-ends*.

3.2.2 Desarrollo de adaptadores

Para permitir el acceso desde la capa EzForge a las herramientas back-end es necesario que éstas dispongan de una API basada en recursos que proporcione la información necesaria a la capa EzForge y con el formato requerido a través de un conjunto de operaciones establecidas.

Para las distintas funcionalidades cubiertas en la capa EzForge se definirán tanto los recursos que modelan estas funcionalidades como los recursos que se espera provean las herramientas subyacentes que son las que realmente ofrecen dicha funcionalidad.

El desarrollo de un adaptador para integrar una herramienta en EzForge consiste, por tanto, en diseñar e implementar una capa REST con los recursos y operaciones requeridos por la capa EzForge, de forma que los manejadores de estos recursos se encargan de acceder a las herramientas back-end a través de alguno de los modelos de acceso a la información comentados en el punto anterior, realizan las operaciones necesarias para obtener los datos requeridos y con ello construyen la respuesta en el formato solicitado por la capa EzForge, que suele ser XML.

Al realizar el modelo de datos de EzForge se ha definido que los adaptadores pueden ser de dos tipos: dinámicos y estáticos, aunque estos no son excluyentes, es decir, un adaptador puede ser a la vez dinámico y estático.

- **Adaptadores estáticos:** son aquellos adaptadores que están realizados expresamente para una herramienta y una instancia de la misma concreta. Es decir, si tenemos una instalación de la misma herramienta en otro lugar, este adaptador no serviría, sería necesario hacer algún cambio para indicar el nuevo lugar donde se encuentra la instancia.
- **Adaptadores dinámicos:** son adaptadores realizados de forma que pueden usarse para las distintas instalaciones de una herramienta en máquinas diferentes, para ello las URIs de sus recursos incluyen la URL de la instancia concreta.
- **Adaptadores que cubren ambos tipos:** estos adaptadores son adaptadores dinámicos pero que además están desarrollados para acceder a una instancia de herramienta concreta por defecto, es decir, si la URI que invoca al adaptador contiene una URL, se accederá a la

herramienta existente en dicha URL, pero si no contiene nada se accederá a la que está configurada por defecto.

Posteriormente, cuando desde el cliente se va a usar una u otra herramienta, en la capa EzForge se anotará la URI del adaptador que se va a usar para dicha herramienta, de esta forma cada vez que se necesite algo de la funcionalidad que proporciona, se hará una petición al recurso correspondiente de ese adaptador (añadiendo la URL de la herramienta en la URI si el adaptador fuera dinámico).

3.2.3 Integración de GForge

EzForge proporciona la funcionalidad de gestión de proyectos, pero además permite al usuario ver los proyectos que tenga en otros repositorios externos o herramientas externas de gestión de proyectos. Por ahora se ha desarrollado un adaptador para GForge, de forma que se puedan ver los proyectos a los que un usuario pertenece en una forja basada en GForge.

Este adaptador utiliza la API SOAP que proporciona GForge para acceder a su información (comentar que esta API no está completa en la versión open source de GForge, aunque proporciona operaciones básicas que cubren la mayoría de las necesidades de integración de EzForge).

El adaptador que se ha desarrollado para -Forge es a la vez estático y dinámico, por lo que si no se indica ninguna URL de instancia de GForge en la URI de invocación del adaptador, la información será obtenida de la forja de la Comunidad de Software Libre MORFEO, que es la que está configurada por defecto en el adaptador. Debido a esto y a que la integración de GForge con EzForge puede realizarse sin necesidad del consentimiento de los mantenedores del sitio, cualquier servicio G-Forge cuya API SOAP se encuentre activa puede ser integrado en el espacio de trabajo de los usuarios.

Se puede encontrar una descripción detallada de la API SOAP de G-Forge en el Anexo.

3.2.4 Integración de MediaWiki

Se ha identificado también para EzForge la funcionalidad de wiki como una funcionalidad interesante y habitual que es bastante útil para el desarrollo de proyectos en colaboración, por lo que se ha decidido incluir esta funcionalidad en el núcleo, para lo cual se han definido e implementado los recursos que modelan la funcionalidad de wiki.

Para proporcionar esta funcionalidad, por el momento se ha elegido la herramienta MediaWiki, que ofrece una API en PHP que nos ha permitido el acceso a su información (comentar que esta API está en desarrollo y aún no proporciona la funcionalidad completa). Haciendo uso de esta API se ha desarrollado el adaptador REST que permitirá que se pueda acceder a los contenidos existentes en una wiki basada en MediaWiki. Este adaptador es dinámico, por lo que puede ser utilizado para distintas wikis basadas en MediaWiki indicando la URL de la wiki correspondiente.

3.2.5 Integración de servicios ad-hoc

Este apartado contempla la incorporación de herramientas en EzForge que se desarrollan expresamente para ser integradas en esta forja. En estos casos no es necesario disponer de un adaptador para el acceso a esta herramienta, sino que la propia herramienta puede desarrollarse con un modelo orientado a recursos siguiendo para el diseño de estos recursos lo especificado para su integración en EzForge.

3.3 Núcleo de servicios

El núcleo de servicios de EzForge implementa la funcionalidad necesaria en el propio núcleo y que no es proporcionada por ninguna herramienta de back-end, como puede ser la gestión de los usuarios de la forja, la infraestructura de autenticación y la gestión de las funcionalidades y herramientas integradas en la forja. Este núcleo de servicios es, por tanto, el responsable de proporcionar la funcionalidad necesaria para garantizar la lógica de integración de nuevas funcionalidades y herramientas.

3.3.1 Gestión de usuarios

Es necesario realizar desde el núcleo de la forja una gestión de los usuarios que están registrados en la misma. Este sistema de gestión de usuarios permite dar de alta nuevas cuentas de usuario, modificar la información de las mismas y asignar a los usuarios un rol determinado que les permitirá llevar a cabo unas operaciones u otras.

Los roles podrán además ser definidos en base a permisos. Cada rol se compone de una lista de permisos cuyo valor indicará qué le está permitido realizar al usuario para una determinada acción y qué no.

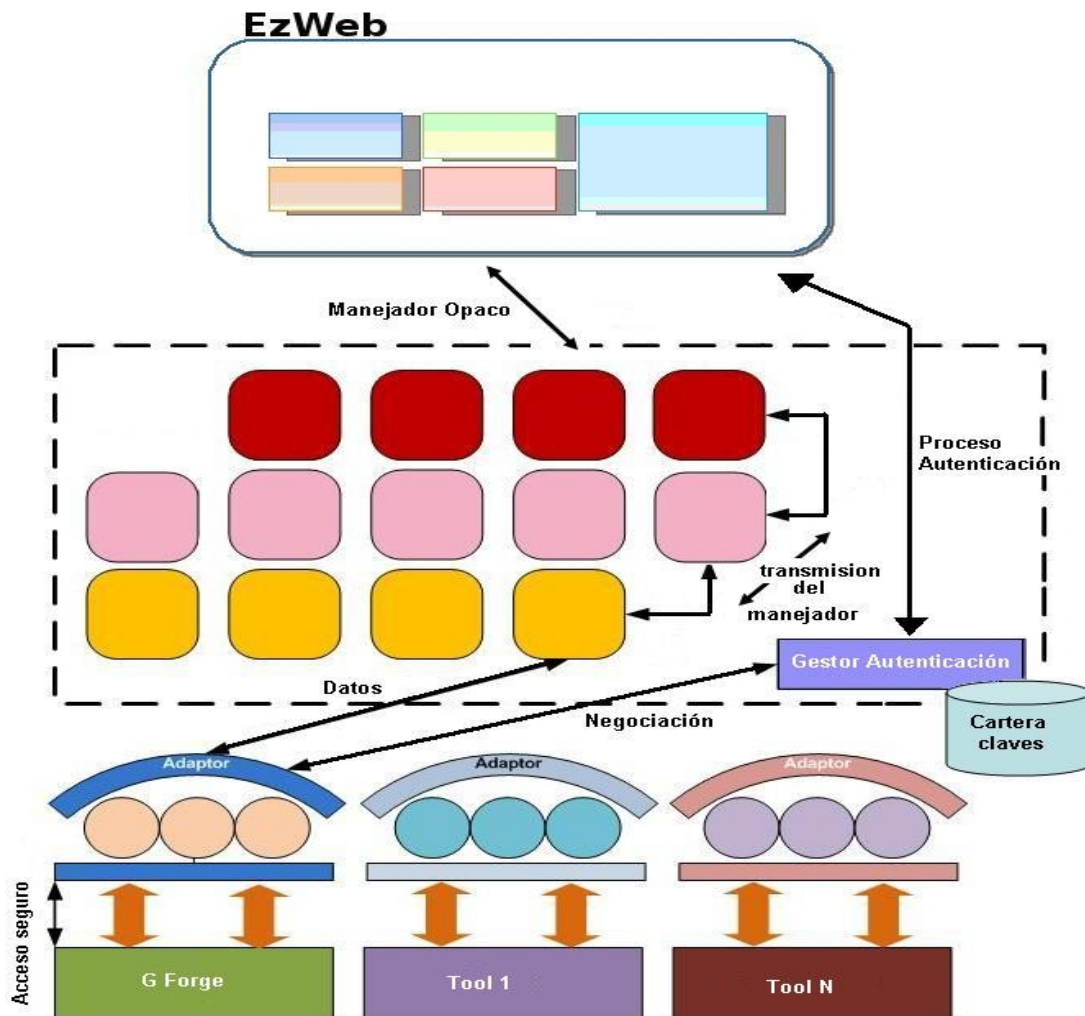
Se proporcionan un conjunto de recursos (indicados en el apartado 3.1 Capa de recursos) para poder realizar todas las operaciones relacionadas con la gestión de usuarios, roles y permisos.

3.3.2 Autenticación

El núcleo de la forja debe proporcionar un sistema para la autenticación de los usuarios en la misma, así como en las herramientas subyacentes.

Cuando se integra una herramienta en la forja se pueden dar dos casos: que se integre una herramienta ya existente que disponga de su propio sistema de autenticación, o que se integre una herramienta desarrollada a medida para la forja que use la infraestructura de autenticación proporcionada por la misma.

Para dar soporte a esta autenticación se ha planteado la infraestructura de autenticación que muestra la siguiente figura:



Esta infraestructura de autenticación consta de dos partes principales: una que permite la autenticación de usuarios en la forja, y otra que permitirá autenticar a los usuarios en las herramientas subyacentes de forma que se pueda hacer uso de la funcionalidad proporcionada por las mismas.

Para la autenticación del usuario en la forja se prevee proporcionar dos métodos, uno basado en autenticación común con nombre de usuario y contraseña, y otro basado en OpenId. Una vez que el usuario se autentique en la forja se le proporcionará al cliente un manejador opaco que le identificará unívocamente. Este manejador se irá transmitiendo a través de los recursos de la capa EzForge hasta llegar a los adaptadores de las herramientas integradas, los cuales lo usarán para autenticar al usuario en dicha herramienta.

La autenticación en las herramientas subyacentes se apoya en este manejador opaco y también en otro artefacto de la capa EzForge denominado *cartera de claves*, el cual contiene para cada usuario de la forja el nombre de usuario y password que tiene el mismo en cada una de las herramientas que utiliza. De esta forma, los adaptadores de las herramientas solicitarán al *Gestor de Autenticación* que le proporcione el nombre de cuenta y contraseña para su

herramienta correspondiente al usuario identificado por el manejador opaco que le ha llegado al adaptador.

3.3.3 Gestión de funcionalidades y herramientas

La capa EzForge necesita tener conocimiento de las funcionalidades que proporciona, las herramientas que tiene integradas, los adaptadores a utilizar para cada herramienta. Para ello, en esta capa de recursos genéricos se mantiene una base de datos en cuyas tablas se almacenan las funcionalidades que se ofrecen, las herramientas que se han integrado junto con las funcionalidades que ofrece cada una, y los adaptadores disponibles para el acceso a estas herramientas.

Se proporciona un conjunto de recursos para que los clientes puedan consultar las funcionalidades proporcionadas por la forja, puedan ver las herramientas que cubren dichas funcionalidades y seleccionar la que quieran usar, elegir el adaptador que deseen para el acceso a la herramienta.

ANEXO : API SOAP DE GFORGE

- **login()**: permite iniciar sesión en el sistema.
 - Entradas:
 - `userid` : `xsd:string` → nombre de usuario
 - `passwd` : `xsd:string` → contraseña
 - Salidas:
 - `loginResponse` : `xsd:string` → identificador de sesión
- **logout()**: permite cerrar una sesión previamente abierta en el sistema.
 - Entradas:
 - `session_ser` : `xsd:string` → identificador de sesión
 - Salidas:
 - `logoutResponse` : `xsd:string` → respuesta de la operación
- **getGroups()**: obtiene la información de los proyectos indicados en `group_ids`
 - Entradas:
 - `session_ser` : `xsd:string` → identificador de sesión
 - `group_ids` : `tns:ArrayOfInt` → colección de identificadores de proyectos
 - Salidas:
 - `returns` : `tns:ArrayOfGroup` → colección de datos de proyectos
- **getGroupsByName()**: obtiene la información de los proyectos indicados en `group_names`
 - Entradas:
 - `session_ser` : `xsd:string` → identificador de sesión
 - `group_names` : `tns:ArrayOfString` → colección de nombres de proyectos
 - Salidas:
 - `returns` : `tns:ArrayOfGroup` → colección de datos de proyectos
- **getPublicProjectNames()**: obtiene los nombres de todos los proyectos públicos.
 - Entradas:
 - `session_ser` : `xsd:string` → identificador de sesión
 - Salidas:
 - `returns` : `tns:ArrayOfString` → colección de nombres de proyectos

- **getUsers():** obtiene la información de los usuarios indicados en user_ids
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - user_ids : tns:ArrayOfInt → colección de identificadores de usuario
 - Salidas:
 - userResponse : tns:ArrayOfUser → colección de datos de usuarios
- **getUsersByName():** obtiene la información de los proyectos indicados en user_names
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - user_names : tns:ArrayOfString → colección de nombres de usuarios
 - Salidas:
 - userResponse : tns:ArrayOfUser → colección de datos de usuarios
- **userGetGroups():** obtiene la información de los proyectos a los que pertenece el usuario user_id
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - user_id : xsd:int → identificador de usuario
 - Salidas:
 - groupResponse : tns:ArrayOfGroup → colección de datos de proyectos
- **getArtifactTypes():** obtiene los tipos de artefacto del proyecto referenciado por group_id
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - Salidas:
 - getArtifactsTypesResponse : tns:ArrayOfArtifactType → colección de datos de tipo de artefacto
- **getArtifacts():** obtiene los artefactos correspondientes al proyecto, tipo, usuario asignado y estado señalados mediante group_id, group_artifact_id, assigned_to y status, respectivamente.
 - Entradas:

- session_ser : xsd:string → identificador de sesión
- group_id : xsd:int → identificador de proyecto
- group_artifact_id : xsd:int → identificador del tipo de artefacto
- assigned_to : xsd:int → identificador de usuario al que se asigna el artefacto
- status : xsd:int → estado del artefacto
- Salidas:
 - getArtifactsResponse : tns:ArrayOfArtifact → colección de datos de artefacto
- **addArtifact()**: añade un nuevo artefacto.
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - group_artifact_id : xsd:int → identificador de tipo de artefacto
 - status_id : xsd:int → identificador de estado del artefacto
 - priority : xsd:int → prioridad del artefacto
 - assigned_to : xsd:int → identificador del usuario al que se asigna el artefacto
 - summary : xsd:string → resumen del artefacto
 - details : xsd:string → descripción detallada del artefacto
 - extra_fields : tns:ArrayOfArtifactExtraFieldsData → campos de información extra
 - Salidas:
 - addArtifactsResponse : xsd:int → código de respuesta de la operación
- **updateArtifact()**: actualiza un artefacto existente.
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - group_artifact_id : xsd:int → identificador de tipo de artefacto
 - status_id : xsd:int → identificador de estado del artefacto
 - priority : xsd:int → prioridad del artefacto
 - assigned_to : xsd:int → identificador del usuario al que se asigna el artefacto
 - summary : xsd:string → resumen del artefacto

- details : xsd:string → descripción detallada del artefacto
- extra_fields : tns:ArrayOfArtifactExtraFieldsData → campos de información extra
- Salidas:
 - addArtifactsResponse : xsd:int → código de respuesta de la operación
- **getArtifactFiles()**: obtiene los ficheros asociados a un artefacto
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - group_artifact_id : xsd:int → identificador de tipo de artefacto
 - artifact_id : xsd:int → identificador del artefacto
 - Salidas:
 - addArtifactFilesResponse : tns:ArrayOfArtifactFile → colección de ficheros de artefacto
- **getArtifactFileData()**: obtiene los datos de un fichero de un artefacto
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - group_artifact_id : xsd:int → identificador de tipo de artefacto
 - artifact_id : xsd:int → identificador de estado del artefacto
 - file_id : xsd:int → prioridad del artefacto
 - Salidas:
 - addArtifactFileDataResponse : xsd:string → datos del fichero
- **addArtifactFile()**: crea un nuevo fichero de artefacto
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - group_artifact_id : xsd:int → identificador de tipo de artefacto
 - artifact_id : xsd:int → identificador del artefacto
 - base64_contents : xsd:string → contenido del fichero en base64
 - filename : xsd:string → nombre del fichero
 - filetype : xsd:string → tipo del fichero
 - Salidas:

- addArtifactFileResponse : xsd:int → identificador del fichero creado
- **artifactFileDelete()**: actualiza un artefacto existente.
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - group_artifact_id : xsd:int → identificador de tipo de artefacto
 - artifact_id : xsd:int → identificador del artefacto
 - file_id : xsd:int → identificador del fichero
 - Salidas:
 - artifactFileDeleteResponse : xsd:bool → código de respuesta de la operación
- **getArtifactMessages()**: obtiene los mensajes asociados a un artefacto
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - group_artifact_id : xsd:int → identificador de tipo de artefacto
 - artifact_id : xsd:int → identificador del artefacto
 - Salidas:
 - addArtifactMessagesResponse : tns:ArrayOfArtifactMessage → colección de mensajes de artefacto
- **addArtifactMessage()**: crea un nuevo mensaje de artefacto
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - group_artifact_id : xsd:int → identificador de tipo de artefacto
 - artifact_id : xsd:int → identificador de artefacto
 - body : xsd:string → cuerpo del mensaje
 - Salidas:
 - addArtifactMessagesResponse : xsd:int → identificador del mensaje creado
- **getArtifactTechnicians()**: obtiene la lista de usuarios técnicos de un artefacto
 - Entradas:
 - session_ser : xsd:string → identificador de sesión

- group_id : xsd:int → identificador de proyecto
- group_artifact_id : xsd:int → identificador de tipo de artefacto
- Salidas:
 - addArtifactTechniciansResponse : tns:ArrayOfUser → colección de usuarios técnicos del artefacto
- **artifactSetMonitor()**: establece un monitor para un artefacto
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - group_artifact_id : xsd:int → identificador de tipo de artefacto
 - artifact_id : xsd:int → identificador del artefacto
 - Salidas:
 - artifactSetMonitorResponse : xsd:bool → resultado de la operación
- **artifactIsMonitoring()**: comprueba si se encuentra activa la monitorización de un artefacto
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - group_artifact_id : xsd:int → identificador de tipo de artefacto
 - artifact_id : xsd:int → identificador del artefacto
 - Salidas:
 - artifactIsMonitoringResponse : xsd:bool → estado de la monitorización
- **artifactDelete()**: elimina un artefacto
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - group_artifact_id : xsd:int → identificador de tipo de artefacto
 - artifact_id : xsd:int → identificador del artefacto
 - Salidas:
 - artifactDeleteResponse : xsd:bool → resultado de la operación
- **artifactTypeIsMonitoring()**: comprueba si se encuentra activa la monitorización de un tipo de artefacto
 - Entradas:

- session_ser : xsd:string → identificador de sesión
- group_id : xsd:int → identificador de proyecto
- group_artifact_id : xsd:int → identificador de tipo de artefacto
- Salidas:
 - artifactTypesMonitoringResponse : xsd:bool → estado de la monitorización
- **artifactGetChangeLog():** obtiene la información de modificaciones del artefacto
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - group_artifact_id : xsd:int → identificador de tipo de artefacto
 - artifact_id : xsd:int → identificador del artefacto
 - Salidas:
 - artifactGetChangeLogResponse : tns:ArrayOfArtifactChangeLog → colección de cambios sobre el artefacto
- **artifactGetViews():** obtiene la información de las vistas del tipo de artefacto
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - group_artifact_id : xsd:int → identificador de tipo de artefacto
 - Salidas:
 - getArtifactTypesResponse : tns:ArrayOfArtifactQuery → colección de vistas sobre el tipo de artefacto
- **artifactDeleteView():** elimina una vista vistas del tipo de artefacto
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - group_artifact_id : xsd:int → identificador de tipo de artefacto
 - artifact_query_id : xsd:int → identificador de consulta de artefacto
 - Salidas:
 - artifactDeleteViewResponse : xsd:bool → resultado de la operación
- **artifactSetView():** establece una vista para un artefacto

- Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - group_artifact_id : xsd:int → identificador de tipo de artefacto
 - artifact_query_id : xsd:int → identificador de consulta de artefacto
- Salidas:
 - artifactSetViewResponse : xsd:bool → resultado de la operación
- **artifactCreateView()**: crea una vista para un artefacto
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - group_artifact_id : xsd:int → identificador de tipo de artefacto
 - artifact_query_id : xsd:int → identificador de consulta de artefacto
 - name : xsd:string → nombre de la vista
 - status : xsd:int → identificador de estado
 - assignee : tns:ArrayOfUserID → colección de usuarios a los que está asignada
 - moddaterange : xsd:string
 - sort_col : xsd:string
 - sort_ord : xsd:string
 - extra_fields : tns:ArrayOfArtifactExtraFieldsData
 - opendaterange : xsd:string
 - closedaterange : xsd:string
 - Salidas:
 - artifactCreateViewResponse : xsd:int → identificador de la vista creada
- **artifactUpdateView()**: actualiza una vista de un artefacto
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - group_artifact_id : xsd:int → identificador de tipo de artefacto
 - artifact_query_id : xsd:int → identificador de consulta de artefacto
 - name : xsd:string → nombre de la vista
 - status : xsd:int → identificador de estado

- assignee : tns:ArrayOfUserID → colección de usuarios a los que está asignada
- moddaterange : xsd:string
- sort_col : xsd:string
- sort_ord : xsd:string
- extra_fields : tns:ArrayOfArtifactExtraFieldsData
- opendaterange : xsd:string
- closedaterange : xsd:string
- Salidas:
 - artifactCreateViewResponse : xsd:int → identificador de la vista creada
- **getProjectGroups():** obtiene la lista de grupos de un proyecto
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - Salidas:
 - getProjectGroupsResponse : tns:ArrayOfProjectGroup → colección de grupos de proyecto
- **getProjectTasks():** obtiene la lista de tareas de un proyecto
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - group_project_id : xsd:int → identificador de relacion entre grupo y proyecto
 - assigned_to : xsd:int → identificador de usuario asignado
 - status : xsd:int → identificador de estado de la tarea
 - category : xsd:int → identificador de categoría de la tarea
 - group : xsd:int → identificador de grupo de proyecto
 - Salidas:
 - getProjectTasksResponse : tns:ArrayOfProjectTask → colección de tareas
- **addProjectTask():** crea una nueva tarea en un proyecto
 - Entradas:
 - session_ser: xsd:string → identificador de sesión
 - group_id: xsd:int → identificador de proyecto

- group_project_id: xsd:int → identificador de relación entre grupo y proyecto
- summary: xsd:string → resúmen de la tarea
- details: xsd:string → descripción de la tarea
- priority: xsd:int → prioridad de la tarea
- hours: xsd:int → estimación de horas que durará la tarea
- start_date: xsd:int → fecha de inicio de la tarea
- end_date: xsd:int → fecha de fin de la tarea
- category_id: xsd:int → identificador de categoría de la tarea
- percent_complete: xsd:int → porcentaje de completitud de la tarea
- assigned_to: tns:ArrayOfint → colección de identificadores de usuarios asignados a la tarea
- dependent_on: tns:ArrayOfint → colección de identificadores de tareas dependientes de la misma
- duration: xsd:int → duración de la tarea
- parent_id: xsd:int → identificador de la tarea maestra
- Salidas:
 - addProjectTaskResponse : xsd:int → identificador de la tarea creada
- **updateProjectTask()**: actualiza una tarea existente en un proyecto
 - Entradas:
 - session_ser: xsd:string → identificador de sesión
 - group_id: xsd:int → identificador de proyecto
 - group_project_id: xsd:int → identificador de relación entre grupo y proyecto
 - summary: xsd:string → resúmen de la tarea
 - details: xsd:string → descripción de la tarea
 - priority: xsd:int → prioridad de la tarea
 - hours: xsd:int → estimación de horas que durará la tarea
 - start_date: xsd:int → fecha de inicio de la tarea
 - end_date: xsd:int → fecha de fin de la tarea
 - category_id: xsd:int → identificador de categoría de la tarea
 - percent_complete: xsd:int → porcentaje de completitud de la tarea
 - assigned_to: tns:ArrayOfint → colección de identificadores de usuarios asignados a la tarea

- dependent_on: tns:ArrayOfint → colección de identificadores de tareas dependientes de la misma
- duration: xsd:int → duración de la tarea
- parent_id: xsd:int → identificador de la tarea maestra
- Salidas:
 - updateProjectTaskResponse : xsd:int → identificador de la tarea actualizada
- **getProjectTaskCategories():** obtiene la lista de categorías de tarea de un proyecto
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - group_project_id : xsd:int → identificador de relacion entre grupo y proyecto
 - Salidas:
 - getProjectTaskCategoriesResponse : tns:ArrayOfProjectCategory → colección de categorías de tarea
- **getProjectMessages():** obtiene la lista de comentarios de una tarea en un proyecto
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - group_project_id : xsd:int → identificador de relacion entre grupo y proyecto
 - project_task_id : xsd:int → identificador de tarea
 - Salidas:
 - getProjectMessagesResponse : tns:ArrayOfProjectMessage → colección de comentarios de tarea
- **addProjectMessage():** crea un nuevo comentario de una tarea en un proyecto
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - group_project_id : xsd:int → identificador de relacion entre grupo y proyecto
 - project_task_id : xsd:int → identificador de tarea
 - body : xsd:string → cuerpo del comentario

- Salidas:
 - addProjectMessageResponse : xsd:int → identificador del comentario creado
- **getProjectTechnicians():** obtiene la lista de usuarios técnicos de un proyecto
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - group_project_id : xsd:int → identificador de relacion entre grupo y proyecto
 - Salidas:
 - getProjectTechniciansResponse : tns:ArrayOfUser → colección de usuarios técnicos de un proyecto
- **getPackages():** obtiene la lista de paquetes de distribución de un proyecto
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - Salidas:
 - getPackagesResponse : tns:ArrayOfFRSPackage → colección de usuarios técnicos de un proyecto
- **addPackage():** crea un nuevo paquete de distribución de un proyecto
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - package_name : xsd:string → nombre del paquete
 - is_public : xsd:int → condición de público del paquete
 - Salidas:
 - addPackageResponse : xsd:int → identificador del paquete creado
- **addPackage():** crea un nuevo paquete de distribución de un proyecto
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - package_name : xsd:string → nombre del paquete
 - is_public : xsd:int → condición de público del paquete

- Salidas:
 - addPackageResponse : xsd:int → identificador del paquete creado
- **getReleases()**: obtiene la lista de publicaciones de un paquete de distribución de un proyecto
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - package_id : xsd:int → identificador del paquete
 - Salidas:
 - getReleasesResponse : tns:ArrayOfFRSRelease → colección de identificadores de las publicaciones
- **addRelease()**: crea una nueva publicación de un paquete de distribución de un proyecto
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - package_id : xsd:int → identificador del paquete
 - name : xsd:string → nombre de la publicación
 - notes : xsd:string → notas sobre la publicación
 - changes : xsd:string → cambios en la publicación
 - release_date : xsd:int → fecha de publicación
 - Salidas:
 - addReleaseResponse : xsd:int → identificador de la publicación creada
- **getFiles()**: obtiene la lista ficheros de una publicación de un paquete de distribución dentro de un proyecto
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - package_id : xsd:int → identificador del paquete
 - release_id : xsd:int → identificador de la publicación
 - Salidas:
 - getFilesResponse : tns:ArrayOfFRSFile → colección de identificadores de ficheros

- getFile(): obtiene el contenido de un fichero de una publicación de un paquete de distribución dentro de un proyecto
- Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - package_id : xsd:int → identificador del paquete
 - release_id : xsd:int → identificador de la publicación
 - file_id : xsd:int → identificador del fichero
- Salidas:
 - getFileResponse : xsd:string → contenido del fichero
 - addFile(): crea un nuevo fichero de una publicación de un paquete de distribución dentro de un proyecto
- Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - package_id : xsd:int → identificador del paquete
 - release_id : xsd:int → identificador de la publicación
 - name : xsd:string → nombre del fichero
 - base64_contents : xsd:string → contenido del fichero en base64
 - type_id : xsd:int → identificador del tipo de fichero
 - processor_id : xsd:int → identificador de procesador
 - release_time : xsd:int → fecha de publicación
- Salidas:
 - addFileResponse : xsd:string → estado de la respuesta
- **getDocumentStates():** obtiene los estados de los documentos
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - Salidas:
 - getDocumentStatesResponse : tns:ArrayOfDocumentState → colección de estados de documento
- **getDocumentLanguages():** obtiene los lenguajes de documentos
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - Salidas:

- `getDocumentLanguagesResponse` :
`tns:ArrayOfDocumentLanguage` → colección de lenguajes de documento
- **addDocument()**: crea un nuevo documento en un proyecto
 - Entradas:
 - `session_ser` : `xsd:string` → identificador de sesión
 - `group_id` : `xsd:int` → identificador de proyecto
 - `doc_group` : `xsd:int` → identificador de grupo de documentación
 - `title` : `xsd:string` → título del documento
 - `description` : `xsd:string` → descripción del documento
 - `language_id` : `xsd:int` → identificador del lenguaje del documento
 - `base64_contents` : `xsd:string` → contenido del documento en base64
 - `filename` : `xsd:string` → nombre del fichero
 - `file_url` : `xsd:string` → URL del fichero
 - Salidas:
 - `addDocumentResponse` : `xsd:int` → identificador del documento
- **updateDocument()**: actualiza un documento existente en un proyecto
 - Entradas:
 - `session_ser` : `xsd:string` → identificador de sesión
 - `group_id` : `xsd:int` → identificador de proyecto
 - `doc_group` : `xsd:int` → identificador de grupo de documentación
 - `title` : `xsd:string` → título del documento
 - `description` : `xsd:string` → descripción del documento
 - `language_id` : `xsd:int` → identificador del lenguaje del documento
 - `base64_contents` : `xsd:string` → contenido del documento en base64
 - `filename` : `xsd:string` → nombre del fichero
 - `file_url` : `xsd:string` → URL del fichero
 - `state_id` : `xsd:int` → identificador del estado del documento
 - Salidas:
 - `updateDocumentResponse` : `xsd:int` → identificador del documento
- **addDocumentGroup()**: crea un nuevo grupo de documento en un proyecto

- Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - groupname : xsd:string → nombre del grupo
 - parent_doc_group : xsd:int → identificador del grupo padre
- Salidas:
 - addDocumentGroupResponse : xsd:int → identificador del documento
- **updateDocumentGroup()**: crea un nuevo grupo de documento en un proyecto
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - new_groupname : xsd:string → nombre del grupo
 - new_parent_doc_group : xsd:int → identificador del grupo padre
 - Salidas:
 - updateDocumentGroupResponse : xsd:int → identificador del documento
- **getDocuments()**: obtiene los documentos de un grupo de documentos en un proyecto
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - doc_group : xsd:int → identificador del grupo
 - Salidas:
 - getDocumentsResponse : tns:ArrayOfDocument → colección de documentos
- **getDocumentGroups()**: obtiene los grupos de documentos de un proyecto
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - Salidas:
 - getDocumentGroupsResponse : tns:ArrayOfDocument → colección de documentos
- **getDocumentGroup()**: obtiene el grupo de documentos de un proyecto

- Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - doc_group : xsd:int → identificador de grupo de documento
- Salidas:
 - getDocumentGroupResponse : tns:DocumentGroup → información de documento
- **getDocumentFiles()**: obtiene los ficheros de un documento perteneciente a un proyecto
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - doc_id : xsd:int → identificador de documento
 - Salidas:
 - getDocumentFilesResponse : tns:ArrayOfDocumentFile → colección de ficheros de documento
- **documentDelete()**: elimina un documento de un proyecto
 - Entradas:
 - session_ser : xsd:string → identificador de sesión
 - group_id : xsd:int → identificador de proyecto
 - doc_id : xsd:int → identificador de documento
 - Salidas:
 - documentDeleteResponse : xsd:boolean → estado de la operación

REFERENCIAS

- [1] “Principled Design of the Modern Web Architecture”, *Roy T. Fielding, Richard N. Taylor. ACM Transactions on Internet Technology, Vol. 2, No. 2 2002.*
- [2] “Dealing With Change: Components Versus Services”, *Ahmed Elfatry, Communications of the ACM, Vol.50, No.8 2007*