ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

TRABAJO FIN DE GRADO

SERVICIO DE DIRECTORIO VIRTUAL DEL EDIFICIO DE INGENIERÍA DE TELECOMUNICACIÓN

(VIRTUAL DIRECTORY SERVICE OF TELECOMMUNICATION ENGINEERING BUILDING)

Para acceder al título de

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

AUTOR: RUBÉN ROMANO POVEDA

3-2017

Contenido

Índice de figuras → 4

Resumen → 7

Abstract → 8

1. Introducción → 9

1.1. Motivación → 9
1.2. Objetivos → 9
1.3. Organización del documento → 9

2. Conceptos teóricos → 10

2.1. LDAP → 10
2.2. Servicio WEB → 12
2.2.1. API → 12
2.2.2. Web de acceso → 12
2.2.3. App Android → 12

3. Aspectos prácticos → 13

3.1. Definición del problema → 13
3.2. Escenario de aplicación → 13
3.3. Requerimientos software → 13

3.3.1. Software de virtualización → 13
3.3.2. Sistema operativo del servidor → 13
3.3.3. OpenLDAP → 14
3.3.4. JXplorer → 14
3.3.5. Eclipse → 14
3.3.6. Servicio web → 14
3.3.7. Servidor web → 14

4. Implementación → 15

4.1. Configurar CentOS → 15
4.2. Instalar y configurar servidor LDAP → 15
4.3. Estructura del directorio LDAP → 21
4.4. Control de acceso → 25
4.5. Servicio Rest → 26

4.5.1. Implementación del servicio → 26
4.5.2. Funciones → 29
4.5.2.1. Conexión al servidor → 30
4.5.2.2. Búsqueda → 32
4.5.2.3. Creación de entradas → 33
4.5.2.4. Borrado de entradas → 35
4.5.2.5. Modificación de entradas → 35
4.5.2.6. Importar alumnos → 37
4.5.2.7. Exportar información → 38

4.5.2.8. Importar información → 39
4.5.2.9. Entradas → 39
4.5.3. Instalación del API → 39
4.5.4. Verificación del servicio → 40
4.5.5. Seguridad → 40
4.6. Página web → 41
4.6.1. Seguridad en la Web → 44
4.7. App Android → 47
5. Conclusión → 52
6. Referencias → 53
7. Anexo 1 : Documentación del API Rest → 56
8. Anexo 2 : Tablas equivalencia API-LDAP → 76

Índice de figuras

1. Ejemplo de directorio LDAP y atributos → 10
2. Ejemplo de entrada en el directorio → 11
3. Ejemplo de búsqueda en el directorio → 11
4. Esquema del proyecto → 12
5. Configuración de red del servidor → 15
6. Instalación LDAP → 15
7. Configuración LDAP → 16
8. Archivo de configuración → 17
9. Daemon LDAP → 20
10. Búsqueda en el directorio → 20
11. Esquema de la estructura del directorio → 21
12. Archivo de estructura → 22
13. Estructura vista en JXplorer → 23
14. Ejemplo de ficha de usuario en formato Ldif → 23
15. Ejemplo de ficha de profesor en formato Ldif → 24
16. Ejemplo de ficha de aula en formato Ldif → 24
17. Añadir entradas a directorio → 24
18. Generación de clave de usuario con openssl → 24
19. Ejemplo de ficha de asignatura en formato Ldif → 25
20. Permisos(ACL) de asignaturas → 25
21. Permisos(ACL) del resto del directorio → 26
22. Instalación Java y Tomcat → 26
23. Configuración auto arranque Tomcat → 27
24. Inicio de proyecto en Eclipse → 28
25. Función REST → 29
26. Función conexión al directorio → 30
27. Generación de token → 31
28. Decodificación de token → 31
29. Creación de filtros para la búsqueda → 32
30. Petición de búsqueda → 33
31. Funciones de creación de entradas → 33
32. Funciones de escritura y lectura en un archivo → 34
33. Borrado de entradas → 35
34. Modificación de entradas → 36
35. Petición de modificación → 37
36. Modificación de contraseña → 37
37. Modificación de DN → 37
38. Importación de alumnos → 38
39. Archivo de exportación de información → 38
40. Instalación del API → 40
41. Generación de certificado de seguridad → 40
42. Configuración conexiones seguras → 41
43. Instalación y configuración servidor Apache → 41
44. Instalación PHP → 42
45. Ejemplo de formulario de la página web → 42
46. Ejemplo de creación de json → 42
47. Ejemplo de conexión usando cURL → 43

48. Ejemplo de decodificación de la respuesta del API → 43
49. Ejemplo de flujo de la página web → 43
50. Esquema seguridad en la web → 45
51. Generación de certificado para la web → 45
52. Configuración de seguridad en la web → 46
53. Desglose de recursos de la app Android → 47
54. Interfaz de Android Studio → 48
55. Método de carga de interfaces → 48
56. Asignación de métodos a pulsaciones de botones → 49
57. Configuración de enlaces con apps preinstaladas → 49
58. Ejemplo de Android Manifest → 50
59. Simulación de dispositivo virtual → 50
60. Ejemplo de pantallas de la app Android 51
61. Ejemplo de login en API → 56
62. Ejemplo de búsqueda de personas en API → 57
63. Ejemplo de búsqueda de aulas en API → 57
64. Ejemplo de búsqueda de asignaturas en API → 58
65. Ejemplo de búsqueda de edificios en API → 58
66. Ejemplo de búsqueda de estudios en API → 59
67. Ejemplo de búsqueda de todas las personas del directorio → 59
68. Ejemplo de obtención de ficha de usuario – solo token → 60
69. Ejemplo de obtención de ficha de usuario → 60
70. Ejemplo de obtención de ficha de usuario - solo datos públicos → 61
71. Ejemplo de obtención de datos de aula → 61
72. Ejemplo de obtención de datos de asignatura – profesor → 62
73. Ejemplo de obtención de datos de asignatura – alumno → 63
74. Ejemplo de obtención de datos públicos de asignatura → 63
75. Ejemplo de obtención de datos de edificio → 64
76. Ejemplo de obtención de datos de estudios → 64
77. Tabla de creación de usuario → 65
78. Ejemplo de creación de usuario → 66
79. Tabla de creación de asignatura → 66
80. Ejemplo de creación de asignatura → 67
81. Tabla de creación de aula → 67
82. Ejemplo de creación de aula → 68
83. Tabla de creación de estudios → 68
84. Ejemplo de creación de estudios → 68
85. Tabla de creación de edificio → 69
86. Ejemplo de creación de edificio → 69
87. Ejemplos de eliminación de entradas → 70
88. Ejemplos de modificación de entradas → 72
89. Ejemplo de exportación de entradas → 73
90. Vistazo archivo de exportación → 73
91. Ejemplo de importación de entradas → 74
92. Ejemplo de importación de alumnos → 75
93. Vistazo archivo de importación de alumnos → 75
94. Tabla equivalencias API-LDAP persona → 76
95. Tabla equivalencias API-LDAP asignatura → 77
96. Tabla equivalencias API-LDAP aula → 77

97. Tabla equivalencias API-LDAP estudios → 77
98. Tabla equivalencias API-LDAP edificio → 78

Resumen

El objetivo de este trabajo es desarrollar e implementar una API que facilite el trabajo de crear aplicaciones usando un servicio de directorio LDAP orientado al grado de Ingeniería de tecnologías de información. La finalidad última es que el usuario que vaya a usar este API no precise de conocimientos de LDAP y pueda usarlo como si se tratase de una base de datos cualquiera.

El directorio contendrá información sobre las asignaturas, los alumnos, los profesores y las aulas del grado de Ingeniería de tecnologías de información.

La API hará uso de un servicio web REST que mediante peticiones HTTP nos permitirá crear y consultar los diferentes elementos que formarán el directorio. La información se transmitirá en formato JSON.

Para demostrar el funcionamiento del API realizado se implementará una página web (que realizará las funciones de administración y consulta) y una app móvil Android (que solo servirá para consultar información) que harán uso de este servicio web para explorar y modificar la información del directorio.

Abstract

This project's aim is to develop and implement an API to make program´s using an LDAP directory service containing info about the university. The end goal is to make the use of an LDAP directory, transparent to the user. This way, they don´t need to know anything about LDAP to use it.

The API will be built using a REST web service that will use HTTP requests.The info will be transferred formated in JSON.

To demonstrate the API´s functioning, a web page will be implemented (with administrative and search functions) and a mobile app for Android operating sistem (with search options only) that will use this web service to explore and modify the info of the directory.

1 - Introducción

A continuación se van a explicar los motivos y objetivos fundamentales que hay detrás de este trabajo.

1.1 - Motivación

La mayor parte de los sistemas de información actuales se basan en el uso intensivo de bases de datos. Este trabajo se realiza para explorar y facilitar el trabajo con un servicio de directorio en lugar de una base de datos convencional. Los servicios de directorio pueden sustituir a las bases de datos tradicionales en algunos entornos que no requieran mantener datos de gran tamaño y ofreciendo una alternativa más segura (ya que no se ve afectado por diversos ataques posibles a las bases de datos basadas en SQL). Esto es especialmente interesante cuando se trata de almacenar contraseñas de cuentas de usuario, informaciones sensibles o sujetas a la ley de protección de datos.

1.2 - Objetivos

Los mecanismos tradicionales hacen uso de la librería OpenLDAP con la que se puede trabajar con directorios LDAP pero es una forma ardua que requiere conocimientos del protocolo. El objetivo de este trabajo es crear una serie de métodos que nos permitan interactuar con un servicio de directorio de forma más amigable. Esto no se aplica solo a los usuarios finales, las aplicaciones también deberán poder hacer uso del API.

Los objetivos más concretos pueden enumerarse como:

- Construcción del servicio de directorio LDAP
- Determinar la estructura del directorio
- Determinar y construir las listas de control de acceso para controlar que información pueden ver determinados usuarios.
- Construir un servicio web Rest
- Construir una serie de métodos que mediante el servicio web permitan interactuar con el servidor LDAP
- Construir una página web sencilla que haciendo uso del servicio web permita consultar y modificar el directorio.
- Construir una app para el sistema operativo Android que permita consultar los datos del directorio.

1.3 - Organización del documento

Este documento se estructura en, además de este apartado de introducción, en tres capítulos fundamentales.

En el capítulo 2 se presentan los conceptos teóricos para a continuación, en el capítulo 3, ahondar en los aspectos prácticos fundamentales de LDAP y de las tecnologías asociadas. En el cuarto capítulo se pasa a explicar en detalle el desarrollo del API, la aplicación de gestión y la aplicación de consulta.

Finalmente se realiza una exposición de las principales conclusiones y líneas futuras de desarrollo.

2 - Conceptos teóricos

A continuación se pasa a explicar los principales términos y conceptos que son necesarios para entender este trabajo.

2.1 – LDAP

LDAP (Lightweight Directory Access Protocol) es un protocolo que permite crear, acceder y modificar un servicio de directorio distribuido basado en el protocolo X.500 pero simplificándolo y añadiendo soporte para TCP/IP (TCP en el puerto 389 y SSL en el puerto 636). Actualmente se hace uso de la versión 3 de LDAP.

Los servicios de directorio son similares a las bases de datos pero añadiendo más seguridad y un entorno distribuido. Están diseñados para almacenar pequeñas cantidades de información pudiendo acceder a datos externos por referencia.

La información en un directorio se organiza en entradas. Las entradas son una colección de atributos con tipo y valor que se designan con un nombre distinguido (DN = Distinguished Name) que es único para cada entrada. En la figura 1 podemos ver la estructura del directorio y a la derecha los atributos con su valor correspondiente.

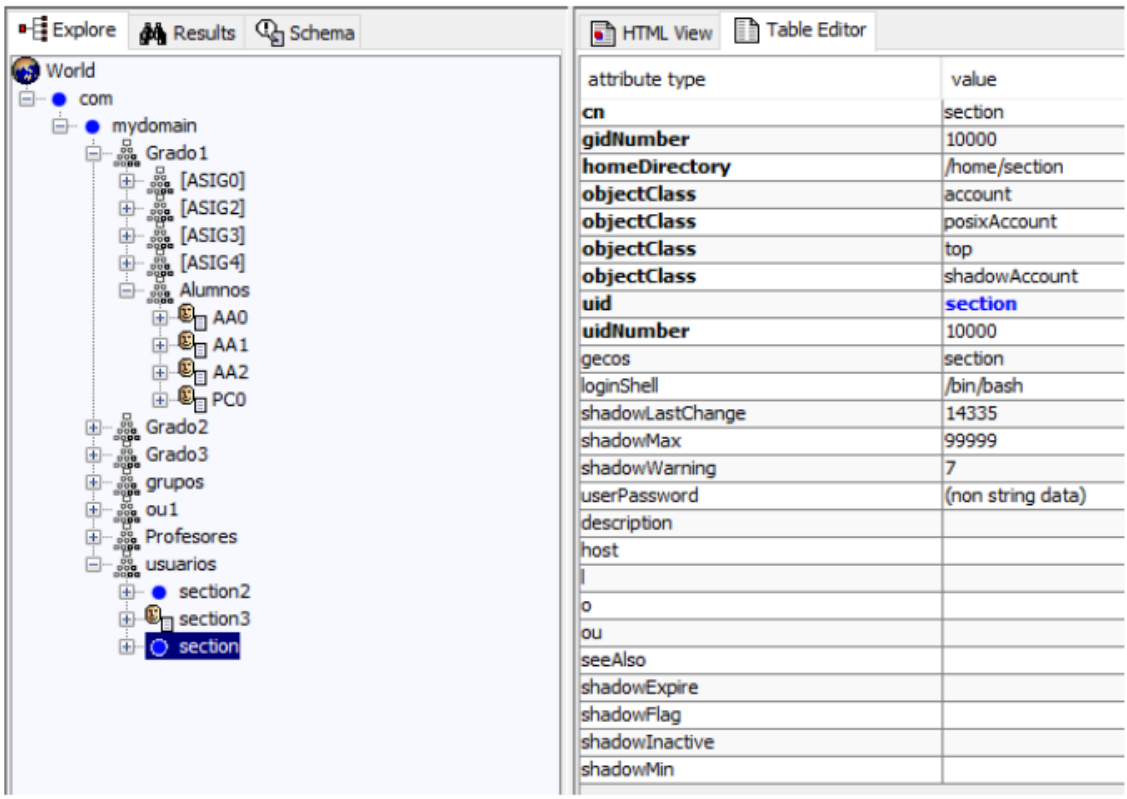

Figura 1: Ejemplo de directorio LDAP y atributos

El ejemplo más común de un servicio de directorio sería el directorio telefónico que consiste en una serie de nombres ordenados alfabéticamente. Cada nombre viene acompañado de una serie de atributos como la dirección o el número de teléfono.

Las entradas en este caso van a representar a los alumnos, profesores, asignaturas y aulas. El nombre distinguido (DN) se forma usando varios RDN's (Relative

Distinguished Name) de forma que nos aseguramos que el DN sea siempre único. El RDN está constituido por algunos atributos de la entrada seguidos del DN de la entrada del padre.

Dada la seguridad del servicio de directorio, es ideal para almacenar credenciales de usuario (usuarios y su password correspondiente) usadas para identificarse en un sistema determinado.

Los esquemas de LDAP son modulares, pudiendo cargar las clases necesarias en cada entrada en función de los atributos que necesitemos. Las clases son colecciones de atributos predefinidos. Podemos cargar las clases que queramos sumándose los atributos de la nueva clase con los de la que ya teníamos.

Las entradas así como los archivos de configuración del directorio se guardan en archivos LDIF como el siguiente:

```
dn: uid=section,ou=usuarios,dc=mydomain,dc=com
uid: section
cn: section
objectClass: account
objectClass: posixAccount
objectClass: top
objectClass: shadowAccount
userPassword: {crypt}$1$TEDFGNB3$1jpJ1Ym4opvpV3orwBznN/
shadowLastChange: 14335
shadowMax: 99999
shadowWarning: 7
loginShell: /bin/bash
uidNumber: 10000
gidNumber: 10000
homeDirectory: /home/section1
gecos: section
```

Figura 2: Ejemplo de entrada en el directorio

Las búsquedas en el directorio se hacen a través de filtros. Podemos construir filtros más complejos con la ayuda de operadores lógicos

```
[root@server ~]# ldapsearch -h localhost -D
"cn=Manager,dc=mydomain,dc=com" -w telematica  -b "dc=mydomain,dc=com"
-s sub "objectclass=*"
```

Figura 3: Ejemplo de búsqueda en el directorio

Las principales operaciones disponibles son:

- Bind/Unbind → conexión/desconexión y autenticación

- Search → búsqueda
- Add/Delete/Modify → añadir, borrar o modificar una entrada

2.2 – Servicio web

En este proyecto vamos a usar un servicio web que hará las funciones de cliente del servidor LDAP ofreciéndonos así una interfaz más amigable con la que interactuar.

Figura 4: Esquema del proyecto

La interacción con el servicio web se hará a través del navegador o la app Android correspondiente.

2.2.1 – API

El API (Application Programming Interface) es el conjunto de métodos creados para interactuar con el servidor LDAP. Hará uso del servicio web REST para intercambiar información entre el usuario y el directorio en formato JSON, información que se usará para alimentar los métodos del API. La información intercambiada no contendrá nada de la sintaxis propia de LDAP para facilitar el uso.

2.2.2 – Web de acceso

Crearemos dos formas de acceder al servicio web, una a través de la app Android diseñada y otra a través de una página web con un diseño sencillo. La página web tendrá las funciones de administración del directorio así como las de consulta haciendo uso de todos los métodos disponibles en el API. Para poder usar la web, habrá que instalar un servidor Apache que la contenga.

2.2.3 – App Android

Como método alternativo de acceso al servicio, crearemos una app para el sistema operativo Android que tendrá solo las funciones de consulta de datos, debido a la limitación que impone el tamaño de pantalla y la usabilidad de una app móvil. El sistema operativo móvil se ha elegido por su facilidad a la hora de programar la app y por el alcance que pueda tener entre los posibles usuarios.

3 – Aspectos prácticos

A continuación se desarrollan las principales actuaciones mediante las cuales se ha llevado a término este trabajo.

3.1 – Definición del problema

Partiendo de la definición del servicio de directorio LDAP, aparece la dificultad de manejar las librerías para adaptarlo a nuestras necesidades. El objetivo último es plantear una estructura de directorio que refleje la estructura de los estudios ofrecidos en la universidad y crear métodos que nos permiten interactuar con ella de forma sencilla. A medida que avancemos iremos viendo que tendremos que definir permisos de acceso para que según que usuario acceda pueda ver o modificar según qué información.

3.2 – Escenario de aplicación

El escenario de aplicación se va a limitar a la información referente al Grado de Ingeniería de Tecnologías de la Información y su funcionalidad se limitará a la de actuar como directorio para consultas.

Sin embargo hay otros proyectos ya realizados que hacen uso de servicios de directorio LDAP y se intentará respetar la estructura para que sigan funcionando esos servicios con los menores cambios posibles. Entre los proyectos ya planteados nos encontramos con funcionalidades planteadas tales como usar el sistema de directorio como método de autentificación en los ordenadores de la universidad.

La API desarrollada en este proyecto no solo estará diseñada para que accedan las personas que usen el servicio sino que también servirá para que otras aplicaciones puedan hacer uso de ello, como se verá en el desarrollo de la app para Android.

3.3 – Requerimientos software

A continuación se indica el software necesario para la realización de las tareas indicadas:

3.3.1 – Software de virtualización

Es necesario un sistema operativo sobre el que montar el servidor LDAP y sobre el que se montará el servicio web. Para no usar distintas máquinas, se crea un sistema virtual utilizando el software de virtualización Oracle VM VirtualBox.

3.3.2 – Sistema operativo del servidor

Para montar el servidor se ha utilizado el sistema operativo CentOS en su versión 6.7.

CentOS es una distribución Linux de código abierto basado en la distribución RedHat.

Para ejecutar este sistema operativo no son necesarios muchos recursos. En este caso y debido a las limitaciones físicas del equipo de ejecución se le han asignado las siguientes características hardware:

- Memoria RAM → 1 GB
- Procesador → En este caso le hemos asignado 1 núcleo y 2 hilos de ejecución

- Almacenamiento → 8 GB
- Red → Adaptador puente

En este equipo lo más importante va a ser la configuración de red ya que se necesita que el cliente y el servidor sean capaces de comunicarse.

3.3.3 – OpenLDAP

OpenLDAP es una librería open source que implementa el protocolo LDAP. Será la piedra sobre la que se montará el servidor LDAP y la que permite interactuar con el servidor directamente. Usa sintaxis y comandos LDAP por lo que no sirve para crear el API pero se podrá usar para crear y configurar el directorio.

3.3.4 – JXplorer

JXplorer es un programa multiplataforma que nos permite acceder y gestionar un servicio de directorio LDAP de forma visual. Sin embargo solo se usará como herramienta de consulta (para ver de forma clara la estructura del directorio por ejemplo). Para añadir las entradas de configuración del directorio se usará un editor de texto como el vi integrado en la consola de Linux.

3.3.5 – Eclipse

Para el desarrollo de las diferentes funciones que formarán la API se usará el entorno de desarrollo Eclipse. Eclipse es un entorno de desarrollo de código abierto usado principalmente para el desarrollo de aplicaciones en lenguaje Java, sin embargo con el paso del tiempo se ha adaptado para usar otros lenguajes como C/C++ o Python. La elección de este entorno de desarrollo se realiza por familiaridad y versatilidad, aunque se podría escoger cualquier otro.

3.3.6 – Servicio WEB

En este proyecto se usará un servicio web REST basado en JSON. REST utiliza el protocolo de transporte HTTP para realizar las peticiones y las respuestas. Como método de codificación de peticiones se usará JSON. La elección de ambas cosas se realiza por rapidez (tanto de envío como de parseo) y por facilidad de desarrollo.

3.3.7 – Servidor web

Se creará un servidor web donde se alojará la página web que nos permitirá usar el servicio. Se ha escogido Apache por ser open source, por su facilidad de implementación y por la cantidad de documentación y ayuda disponible. Apache implementa el protocolo HTTP 1.1.

4 – Implementación

La implementación se ha dividido en diferentes fases en función de los hitos que se deberían cumplir. La primera fase es la configuración de red del equipo que va a realizar las funciones de servidor LDAP

4.1 – Configurar CentOS

Como paso previo a instalar LDAP se ha de configurar el sistema operativo. La única configuración que se debe tocar es la configuración de red. Se deben crear las entradas respectivas en el firewall del sistema para que permita la comunicación con el equipo. Para simplificar las cosas se desactivarán los servicios IPTABLES y SELINUX aunque en una configuración real no se debería hacer.

```
[root@server ~]# service iptables stop
[root@server ~]# service ip6tables stop
[root@server ~]# chkconfig iptables off
[root@server ~]# chkconfig ip6tables off
```

Figura 5: Desactivar IPTables

```
[root@server ~]# vi /etc/selinux/config
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
# enforcing - SELinux security policy is enforced.
# permissive - SELinux prints warnings instead of enforcing.
# disabled - No SELinux policy is loaded.
SELINUX=disabled
# SELINUXTYPE= can take one of these two values:
# targeted - Targeted processes are protected,
# mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

Figura 6: Desactivar SELINUX

Para terminar se reiniciará el servidor para que los cambios tengan efecto.

4.2 – Instalar y configurar LDAP

Una vez tenemos la configuración de red adecuada se procederá a instalar LDAP.

```
[root@server ~]# yum install openldap-servers openldap-clients
```

Figura 7: Instalar OpenLDAP

Cuando se hayan instalado los paquetes necesarios en el sistema, se procederá a configurarlo.

Para inicializar la nueva base de datos debemos copiar un fichero de configuración:

```
[root@server ~]# cp /usr/share/openldap-servers/DB_CONFIG.example
/var/lib/ldap/DB_CONFIG
```

Figura 8: Copiar archivo de configuración

A continuación se deberán dar los permisos necesarios al directorio LDAP

```
[root@server ~]# chown -R ldap:ldap /var/lib/ldap
```

Figura 9: Cambiar la posesión del directorio

Accedemos al directorio LDAP situado en la ruta /etc/openldap y se realiza una copia del directorio slapd.d y del archivo ldap.conf para conservar la configuración original en caso de error.

Se configura el sistema para que el servicio LDAP se ejecute al arrancar la maquina

```
[root@server openldap]# chkconfig slapd on
```

Figura 10: Configuración del arranque automático del demonio de OpenLDAP

Como último paso se escribe la configuración necesaria para nuestro directorio LDAP en el archivo de configuración slapd.conf y ppolicy.ldif

En este archivo será necesario introducir una clave de usuario para acceder al directorio que se generará de la siguiente forma:

```
[root@server openldap]# slappasswd
```

Figura 11: Generar password

Este comando generará un hash a partir de la contraseña que se le pase de argumento.

```
#
# See slapd.conf(5) for details on configuration options.
# This file should NOT be world readable.
#
include /etc/openldap/schema/core.schema
include /etc/openldap/schema/cosine.schema
include /etc/openldap/schema/inetorgperson.schema
include /etc/openldap/schema/nis.schema

# Added for policy
include /etc/openldap/schema/ppolicy.schema

# Allow LDAPv2 client connections.       This is NOT the default.
allow bind_v2

# Do not enable referrals until AFTER you have a working directory
# service AND an understanding of referrals.
#referral       ldap://root.openldap.org

pidfile /var/run/openldap/slapd.pid
argsfile/var/run/openldap/slapd.args

# Load dynamic backend modules:
# modulepath     /usr/lib64/openldap

#Modules available in openldap-servers-overlays RPM package
#Module syncprov.la is now statically linked with slapd and there
#is no need to load it here
#moduleload accesslog.la
#moduleload auditlog.la
#moduleload denyop.la
#moduleload dyngroup.la
#moduleload dynlist.la
#moduleload lastmod.la
#moduleload pcache.la
moduleload ppolicy.la
# moduleload refint.la
# moduleload retcode.la
# moduleload rwm.la
```

Figura 12: Archivo de configuración de LDAP - slapd.conf (1)

```
# moduleload smbk5pwd.la
# moduleload translucent.la
# moduleload unique.la
# moduleload valsort.la

# modules available in openldap-servers-sql RPM package:
# moduleload back_sql.la

# The next three lines allow use of TLS for encrypting connections using a
# dummy test certificate which you can generate by changing to
# /etc/pki/tls/certs, running "make slapd.pem", and fixing permissions on
# slapd.pem so that the ldap user or group can read it. Your client software
# may balk at self-signed certificates, however.
# TLSCACertificateFile /etc/pki/tls/certs/ca-bundle.crt
# TLSCertificateFile /etc/pki/tls/certs/slapd.pem
# TLSCertificateKeyFile /etc/pki/tls/certs/slapd.pem

# Sample security restrictions
# Require integrity protection (prevent hijacking)
# Require 112-bit (3DES or better) encryption for updates
# Require 63-bit encryption for simple bind
# security ssf=1 update_ssf=112 simple_bind=64

# Sample access control policy:
# Root DSE: allow anyone to read it
# Subschema (sub)entry DSE: allow anyone to read it
# Other DSEs:
# Allow self write access
# Allow authenticated users read access
# Allow anonymous users to authenticate
# Directives needed to implement policy:
# access to dn.base="" by * read
# access to dn.base="cn=Subschema" by * read
# access to *
# by self write
# by users read
# by anonymous auth
# if no access controls are present, the default policy
# allows anyone and everyone to read anything but restricts
# updates to rootdn. (e.g., "access to * by * read")
# rootdn can always read and write EVERYTHING!
#######################################################################
#
# ldbm and/or bdb database definitions
#######################################################################
#
```

Figura 13: Archivo de configuración de LDAP - slapd.conf (2)

```
database bdb
suffix  "dc=mydomain,dc=com"
rootdn  "cn=Manager,dc=mydomain,dc=com"
rootpw  {SSHA}j65Y6iumv9ukVjVzE3pZJaVYolr3UXXy

# PPolicy Configuration
overlay ppolicy
ppolicy_default "cn=default,ou=policies,dc=mydomain,dc=com"
ppolicy_use_lockout
ppolicy_hash_cleartext

# The database directory MUST exist prior to running slapd AND
# should only be accessible by the slapd and slap tools.
# Mode 700 recommended.
directory       /var/lib/ldap

# Indices to maintain for this database
index objectClass                               eq,pres
index ou,cn,mail,surname,givenname              eq,pres,sub
index uidNumber,gidNumber,loginShell            eq,pres
index uid,memberUid                             eq,pres,sub
index nisMapName,nisMapEntry                    eq,pres,sub
```

Figura 14: Archivo de configuración de LDAP - slapd.conf (3)

```
dn: ou = policies,dc=mydomain,dc=com
objectClass: organizationalUnit
objectClass: top
ou: policies
# default, policies, example.com
dn: cn=default,ou=policies,dc=mydomain,dc=com
objectClass: top
objectClass: pwdPolicy
objectClass: person
cn: default
sn: dummy value
pwdAttribute: userPassword
pwdMaxAge: 7516800
pwdExpireWarning: 14482463
pwdMinLength: 2
pwdMaxFailure: 10
pwdLockout: TRUE
pwdLockoutDuration: 60
pwdMustChange: FALSE
pwdAllowUserChange: FALSE
pwdSafeModify: FALSE
```

Figura 15: Archivo de configuración ppolicy.ldif

El archivo ppolicy.ldif contiene la configuración de las contraseñas en el directorio con parámetros que indican la duración de las contraseñas o que un usuario no puede cambiar su propia contraseña.

Una vez escritos los dos archivos de configuración, se puede comprobar si están bien escritos con el comando *slaptest.*

Habiendo terminado el proceso de configuración, solo queda iniciar el servidor LDAP:

```
[root@server ~]# service slapd start
```

Figura 16: Arranque del demonio de LDAP

Para comprobar que el servidor está correctamente iniciado y configurado, se realizará una búsqueda:

```
[root@server ~]# ldapsearch -h localhost -D
"cn=Manager,dc=mydomain,dc=com" -w telematica  -b "dc=mydomain,dc=com"
-s sub "objectclass=*"
```

Figura 17: Búsqueda en el directorio

Si todo está correcto, debería devolvernos este resultado:

```
# extended LDIF
#
# LDAPv3
# base <dc=mydomain,dc=com> with scope subtree
# filter: objectclass=*
# requesting: ALL
#

# search result
search: 2
result: 32 No such object

# numResponses: 1
```

Figura 18: Respuesta del servidor LDAP

También se puede comprobar la conexión con el programa JXplorer que conectará pero mostrará varios avisos de error ya que todavía no está configurada la estructura del directorio.

4.3 – Estructura del directorio LDAP

Antes de introducir datos en el directorio, primero hay que definir la estructura que va a tener. La estructura tiene forma de árbol en función de los datos que deberá contener. En este caso deberá contener información sobre alumnos, profesores, asignaturas y aulas. Podrá contener más información pero que no estará presente en el directorio LDAP si no que se accederá a ella por referencia. La estructura queda así:

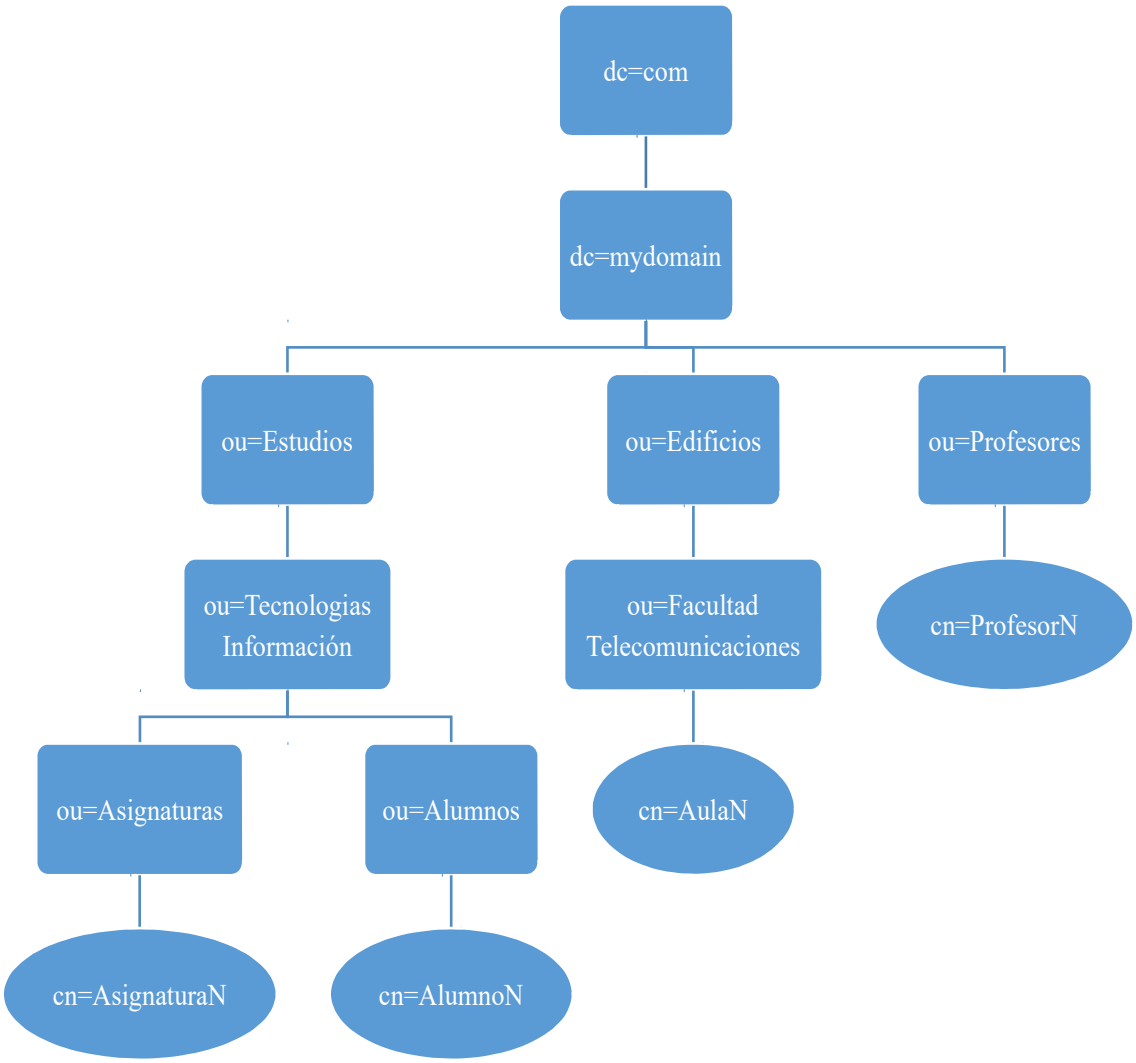

Figura 19: Esquema de la estructura del directorio

- En los directorios LDAP se suele usar el nombre del dominio para los niveles más altos del árbol de tal forma que así se asegura un DN único. Aquellas partes de la estructura del árbol que forman parte del dominio son las dc (domain component).
- Las partes llamadas ou (organizational unit) son entradas usadas para crear nodos del árbol cuya función es reflejar la estructura de la información que contendrá el directorio. En este caso reflejan la estructura de los estudios, limitándose solo al ámbito de la Ingeniería de Tecnologías de la Información.

- Las partes llamadas cn (common name) serán las que contengan la información sobre la que actuaremos (consultas/modificaciones).

Se ha definido esta estructura pensando en ampliarla en el futuro con las diferentes titulaciones y edificios que forman parte de la Universidad de Cantabria.

Para crear esta estructura en el directorio, definiremos un archivo que llamaremos estructura.ldif que contendrá los nodos del árbol:

```
dn: dc=mydomain,dc=com
objectClass: top
objectClass: domain
dc: mydomain

dn: ou=estudios,dc=mydomain,dc=com
objectClass: top
objectClass: OrganizationalUnit
ou: estudios

dn: ou=edificios,dc=mydomain,dc=com
objectClass: top
objectClass: OrganizationalUnit
ou: edificios

dn: ou=profesores,dc=mydomain,dc=com
objectClass:top
objectClass : OrganizationalUnit
ou: profesores

dn: ou=FacultadTelecomunicaciones,ou=edificios,dc=mydomain,dc=com
objectClass:top
objectClass: OrganizationalUnit
ou: FacultadTelecomunicaciones

dn: ou=TecnlogiasInformacion,ou=estudios,dc=mydomain,dc=com
objectClass:top
objectClass:OrganizationalUnit
ou:TecnologiasInformacion

dn: ou=asignaturas,ou=TecnologiasInformacion,ou=estudios,dc=mydomain,dc=com
objectClass:top
objectClass:OrganizationalUnit
ou:asignaturas

dn: ou=alumnos,ou=TecnologiasInformacion,ou=estudios,dc=mydomain,dc=com
objectClass:top
objectClass:OrganizationalUnit
ou:alumnos
```

Figura 20: Estructura del directorio en formato LDIF

Una vez definido el archivo, lo añadiremos al directorio para que tenga efecto con el comando ldapadd:

```
[root@server ~]# ldapadd -x -D "cn=Manager,dc=mydomain,dc=com" -W -f
estructura.ldif
```

Figura 21: Añadir entradas a LDAP

Ahora podemos ver la representación del directorio de forma gráfica con el JXplorer:

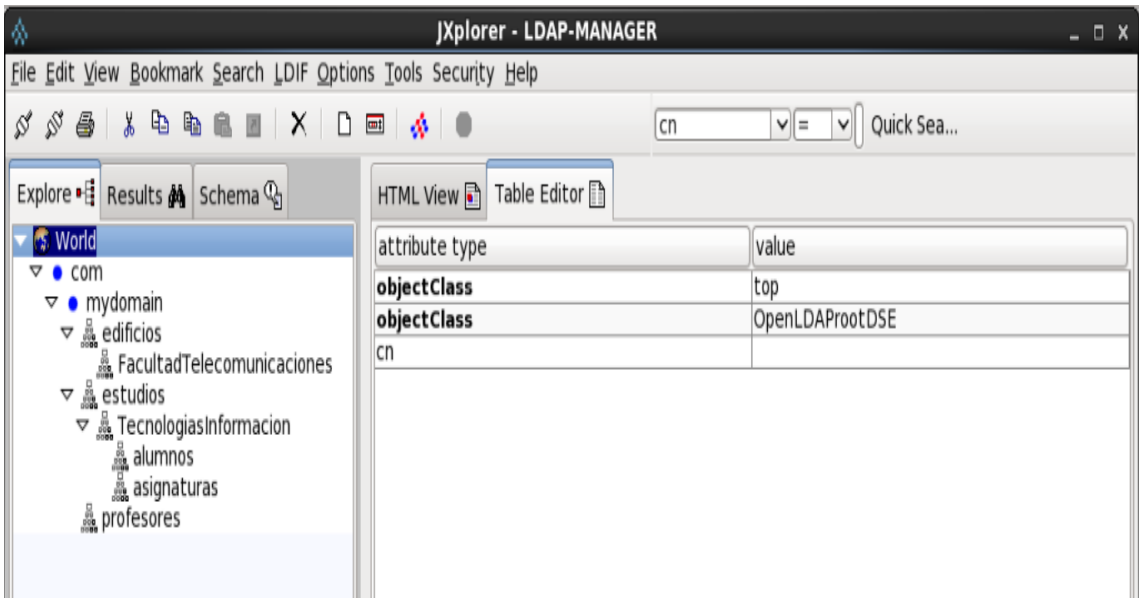

Figura 22: Estructura del directorio vista con JXplorer

Lo siguiente será definir la información que se guardará en cada ficha (alumnos, asignaturas, aulas y profesores).Para cada ficha se creará un fichero ldif con los campos correspondientes:

```
dn: uid=rrp25,ou=alumnos,ou=TecnologiasInformacion,ou=Estudios,dc=mydomain,dc=com
cn: rrp25
uid: rrp25
objectClass: top
objectClass: posixAccount
objectClass: shadowAccount
objectClass: inetOrgPerson
objectClass: person
gn: Ruben
sn: Romano Poveda
mail: rrp25@unican.es
mobile: 622-20-38-44
homePhone: 942-05-62-42
homePostalAddress: Menendez Pelayo 17 1ºC
labeledUri: www.rrp25.alumnos.unican.es
jpegPhoto:
userPassword: {crypt}$1$yxQV55oW$Rd4m2TqME742OQSPwBs/V.
shadowLastChange: 14335
shadowMax: 99999
shadowWarning: 7
loginShell: /bin/bash
homeDirectory: /home/users/rrp25
gecos: rrp25
uidNumber:1
gidNumber:1
```

Figura 23: Ficha de usuario

```
dn: uid=profesor1,ou=Profesores,dc=mydomain,dc=com
cn: profesor1
uid: profesor1
objectClass: top
objectClass: posixAccount
objectClass: shadowAccount
objectClass: inetOrgPerson
objectClass: person
gn: NombreProfesor
sn: Apellidos
mail: profesor@unican.es
mobile: 666-66-66-66
homePhone: 999-99-99-99
telephoneNumber: 777-77-77-77
homePostalAddress: General Davila 5 3ºA
postalAddress: Edificio I+D+i Planta 0
departmentNumber: 108
labeledUri: www.unican.es
jpegPhoto:
userPassword: {crypt}$1$szi1sFGO$nJgq3g6SM5NxAvRfMe5YP1
shadowLastChange: 14335
shadowMax: 99999
shadowWarning: 7
loginShell: /bin/bash
homeDirectory: /home/users/profesor1
gecos: profesor1
uidNumber:2
gidNumber:1
```

Figura 24: Ficha de profesor

Se han mantenido campos relacionados con el trabajo de fin de grado de Alejandro Abascal Crespo (Desarrollo e implementación de una oficina de información virtual) que, si bien no son necesarios para este proyecto en concreto, no estorba y permite que sea compatible.

```
dn: cn=017,ou=Facultad Telecomunicaciones,ou=Edificios,dc=mydomain,dc=com
cn: 017
sn: 017
objectClass: top
objectClass: inetOrgPerson
postalAddress: Planta 0
```

Figura 25: Ficha de aula

Nos falta añadir las asignaturas, pero eso lo veremos en el apartado de control de acceso. Para añadir estas entradas al directorio usaremos el comando ldapadd:

```
[root@server ~]# ldapadd -x -D "cn=Manager,dc=mydomain,dc=com" -W -f
estructura.ldif
```

Figura 26: Añadir entradas a LDAP

Para generar las contraseñas de los usuarios usaremos el comando openssl passwd:

```
# openssl passwd -1 -salt $(openssl rand -base64 6) ThePassword
$1$XpOpurgQ$41bulzoCV8viFy37EX6jk.
```

Figura 27: Generación de la clave de usuario

Usaremos los mismos datos de usuario y password que usamos en el campus virtual.

4.4 – Control de acceso

Para controlar el acceso de según qué información por según que usuario se deben dar varios pasos. El primer paso será crear grupos de usuarios, uno por cada asignatura, de tal forma que formarán parte de ese grupo todos los alumnos y profesores que participen en esa asignatura.

```
dn: cn=Sistemas Operativos,ou=Asignaturas,ou=Tecnologias Informacion,ou=estudios,dc=mydomain,dc=com
cn: Sistemas Operativos
objectClass: top
objectClass: groupOfNames
objectClass: labeledURIObject
labeledURI: www.aulavirtual.com
o: www.guiaacademica.com
member: cn=rrp25,ou=Alumnos,ou=Tecnologias Informacion,ou=Estudios,dc=mydomain,dc=com
owner: cn=profesor1,ou=Profesores,dc=mydomain,dc=com
```

Figura 28: Asignatura.ldif

Para ello se usa la clase groupOfNames que tiene dos argumentos que interesantes: member y owner. De esta forma a los miembros se les dan permisos de lectura y a los poseedores (owner) permisos de escritura.

También se establecen dos enlaces distintos. Un enlace será público y contendrá la guía académica de la asignatura. Otro enlace será privado y contendrá los materiales de la asignatura, que solo podrán ver los que formen parte de ese grupo.

Una vez establecido el grupo y sus miembros hay que otorgarles permiso. Los permisos se otorgan en el archivo de configuración del directorio slpad.conf

Al final del archivo se añaden estas líneas:

```
access to dn.base="cn=Sistemas Operativos,ou=asignaturas,ou=Tecnologias Informacion,ou=estudios,dc=mydomain,dc=com" attrs=o
        by dnattr=owner write
        by * read

access to dn.base="cn=Sistemas Operativos,ou=asignaturas,ou=Tecnologias Informacion,ou=estudios,dc=mydomain,dc=com" attrs=labeledURI
        by dnattr=owner write
        by dnattr=member read
```

Figura 29: Permisos (ACL) para una asignatura

Estas líneas son los permisos de cada asignatura. Al owner se le permite modificar atributos y los miembros solo se les permite leerlos.

El atributo “o” será el que contenga el enlace a la guía académica de forma que todo el mundo pueda verla, mientras que el atributo “labeledUri” contendrá un enlace a los materiales de la asignatura que solo podrán verlo los alumnos de esa asignatura.

Se añaden también estas líneas que se aplican en todo el directorio:

```
access to attrs=userPassword
        by self write
        by anonymous auth
        by * none

access to dn.base=""
        by * read

access to *
        by self write
        by * read
```

Figura 30: Permisos (ACL) para cualquier entrada del directorio

Aquí se está controlando que solo el propio usuario pueda cambiar su contraseña y sus datos y que todos los usuarios puedan acceder al directorio aunque sea en modo lectura.

En esta configuración importa el orden de tal forma que las reglas para las asignaturas van primero, y si el usuario no se rige por ninguna de esas, se aplican estas.

Una vez están todos los cambios en el archivo de configuración, hay que reiniciar el servicio de directorio para que se apliquen los cambios.

```
[root@server ~]# service slapd start
```

Figura 31: Iniciar servidor LDAP

En principio, salvo los materiales propios de cada asignatura, toda la información del directorio será pública. Sin embargo, como no todos los datos son obligatorios, si un usuario quiere hacer público un dato opcional, puede hacerlo sin problema. Si por la contra un usuario quiere que cierta información no sea pública, bastará con dejar ese campo vacío.

4.5 – Servicio Rest

4.5.1 – Implementación del servicio

Se va a desarrollar el servicio Rest en Java usando Eclipse. Antes de empezar a desarrollar el servicio va a ser necesario instalar java, el servidor Tomcat y Eclipse en CentOS. Empezamos por java:

```
[root@server ~]# yum install java-1.7.0-openjdk
```

Figura 32: Instalar Java

Con java ya instalado, toca el turno para Tomcat. Descargamos los binarios de la página web http://tomcat.apache.org/ y los descomprimimos en la carpeta /opt/tomcat (la carpeta no existe así que tendremos que crearla):

```
[root@server ~]# tar xvf apache-tomcat-8*tar.gz -C /opt/tomcat --
strip-components=1
```

Figura 33: Descomprimir e instalar Tomcat

El problema de tomcat es que no se inicia automáticamente, hay que usar los archivos startup.sh y shutdown.sh. Se puede automatizar el arranque del servicio creando el siguiente archivo en /etc/init.d con el nombre tomcat:

```
[root@server ~]# vi /etc/init.d/tomcat
#!/bin/sh
#
# chkconfig: 2345 20 80
# description: Tomcat
#

export TOMCAT_HOME=/opt/tomcat/
start(){
        $TOMCAT_HOME/bin/startup.sh
}
stop(){
        $TOMCAT_HOME/bin/shutdown.sh
}

case "$1" in
        start)
                echo "Iniciando Tomcat..."
                start
        ;;
        stop)
                echo "Deteniendo Tomcat..."
                stop
        ;;
        restart)
                echo "Reiniciando Tomcat..."
                stop
                start
        ;;
        *)
                echo $"Uso:$0{start|stop}"
                exit 1
esac
```

Figura 34: Archivo /etc/init.d/tomcat

```
[root@server ~]# chkconfig tomcat on
```

Figura 35: Configuración de auto inicio de Tomcat

Con tomcat instalado y funcionando, queda instalar Eclipse en CentOS para desarrollar el servicio. Descargamos la última versión desde la página web https://eclipse.org/ .

Una vez instalado se van a necesitar las siguientes librerías:

- Eclipse Java EE Developer Tools
- Eclipse Java Web Developer Tools
- Eclipse Web Developer Tools
- Eclipse XSL Developer Tools
- JST Server Adapters
- JST Server Adapters Extensions
- WAR Products(Incubation)
- WST Server Adapters

Algunas vendrán integradas en Eclipse (depende de la versión que usemos), las que no tengamos se pueden descargar desde Eclipse (Help→Install new software).

Aparte de estas librerías, hay que descargar las siguientes librerías necesarias para ciertas funciones:

- JAX-RS (disponible en el paquete Jersey) – permite usar los métodos propios de un servicio REST
- Nimbus-JOSE – necesaria para usar tokens
- UnboundID – librería que interactuara con el directorio LDAP

Cuando ya está todo preparado, se crea el servicio pulsando File→New→Dinamic Web Project. Hay que cambiar la parte de Target Runtime (seleccionar New Runtime y Apache tomcat v8.0) y la parte de configuración (Modify y añadir Jax-RS).

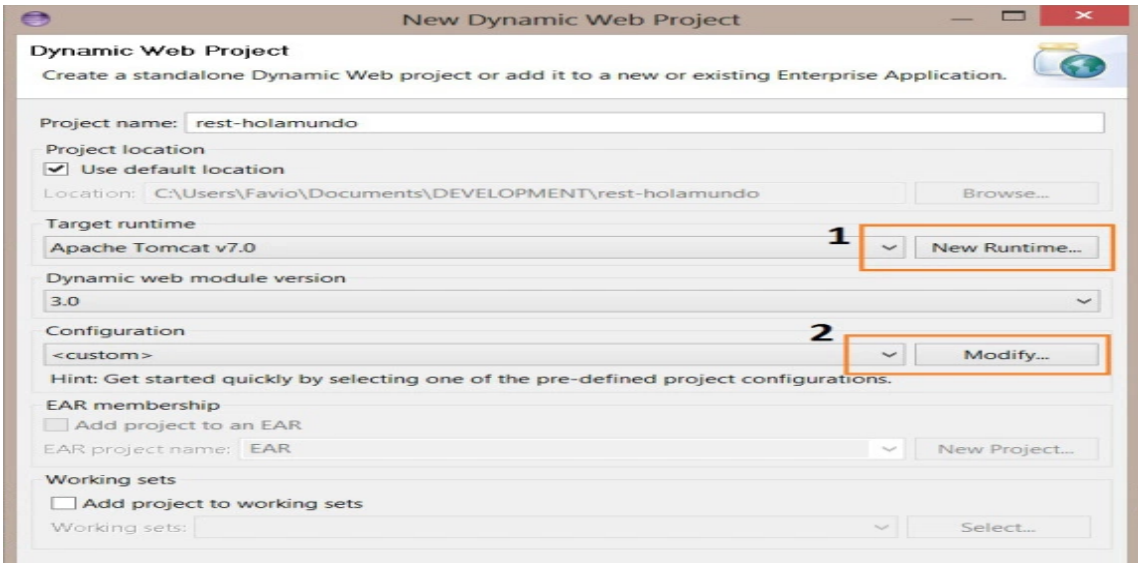

Figura 36: Inicio de proyecto en Eclipse

4.5.2 – Funciones

En este apartado vamos a explicar las funciones desarrolladas para este servicio. Para poder operar con el servidor LDAP usaremos la librería *unboundid* que podremos encontrar en https://www.ldap.com/unboundid-ldap-sdk-for-java. Existen varías librerías que podemos usar para acceder al servidor LDAP y que nos facilitan el desarrollo. Hemos escogido esta librería debido a su sintaxis clara y facilidad de uso.

En este desarrollo usaremos la versión gratuita de la librería por lo que habrá ciertas funciones que no podremos usar.

Aquí se desarrolla una pequeña introducción de las funciones ya que en el anexo se encuentra la documentación necesaria para usar las funciones así como los datos necesarios.

- Búsqueda: está será la función principal del API que nos permitirá buscar datos en el directorio y desde los datos devueltos por la búsqueda, acceder a las entradas correspondientes.
- Entradas: estará disponible una URI para acceder a una entrada directamente (de persona, de asignatura, etc) aportando los datos necesarios para acceder a esa entrada. Nos devolverá toda la información disponible en esa entrada.
- Funciones de gestión de entradas: serán el punto de entrada y gestión de información del directorio. Nos permitirá crear entradas nuevas, modificarlas y borrarlas.
- Funciones de gestión del directorio: nos permitirán realizar backup de los datos del directorio así como restaurar un backup ya creado.
- Funciones especiales: habrá algún caso que requiera una función especial para facilitar las cosas. De momento solo hay una función de este tipo que nos permitirá importar alumnos desde un archivo de texto.

Para implementar la interfaz REST usaremos el método HTTP POST:

```
@POST
@Consumes(MediaType.APPLICATION_JSON)
@Produces("application/json")
public Object PostHTML(JsonEntry input)
```

Figura 37: Función REST

Así luce la definición de una función REST:

@POST indica que usaremos el método HTTP POST

@Consumes indica de que tipo serán los argumentos de entrada, en este caso JSON

@Produces indica de que tipo serán los datos de salida, en este caso JSON

Las indicaciones de un servicio REST correcto indican que no se deberían usar nombres de acciones en las URIS del servicio y que para expresar acciones debemos usar los métodos HTTP GET, POST y PUT, pero en la actualidad los métodos GET y PUT están

en desuso cada vez más y dado que podemos hacerlo todo con el método POST esto nos forzará a usar nombres de acciones en algunas URIS.

Como formato de intercambio de datos hemos escogido JSON, por ser más sencillo de parsear que XML pero se podría haber usado uno u otro sin mayor diferencia.

Como respuesta, el servicio deberá devolver códigos HTTP indicando si la operación ha sido correcta así como los posibles fallos. La lista de errores devueltos se explicará en el anexo.

4.5.2.1 – Conexión al servidor LDAP

El primer paso para poder hacer cualquier acción contra el servidor LDAP será el conectarse a él. Para la conexión usaremos la función *LDAPConnection* de la librería unboundid:

```
conection = new LDAPConnection(ipServer,port);
```

Figura 38: Función de conexión al directorio (1)

Le pasamos como argumento la ip del servidor LDAP y el puerto. Este paso nos realiza la conexión al servidor, pero no la autentificación como usuario del directorio. Para poder acreditarnos como usuario del directorio debemos usar la función *bind* una vez nos hemos conectado.

```
conection.bind(UserDN, password);
```

Figura 39: Función de conexión al directorio (2)

Le pasamos como argumento el DN (Domain Name) de nuestro usuario y nuestra password. La única dificultad de este paso estriba en recuperar el Domain Name que reconstruiremos a partir del usuario.

Si no nos identificamos como usuario del directorio, aun así podemos navegar por el directorio como usuario anónimo por lo que lo único que podremos hacer será consultar la información que sea pública.

Una vez nos hemos identificado en el servidor LDAP, debemos devolverle al usuario del API un token que le identificará como usuario, de forma que no tenga que enviar su usuario y password cada vez que quiera realizar alguna operación. Para ello usaremos la librería Nimbus-JOSE y en concreto la clase *JWSSigner:*

```
 JWSSigner signer = null;
try
{
    signer = new MACSigner(sharedSecret);
}
catch (KeyLengthException e1)
{
    System.out.println("e1 : " + e1);
}

 JWTClaimsSet claimsSet = new JWTClaimsSet.Builder()
         .subject(usuario)
         .claim("password", password)
         .issueTime(new Date())
         .expirationTime(new Date(new Date().getTime() + 3600 * 1000)) // duracion del token = 1 hora
         .build();

 SignedJWT signedJWT = new SignedJWT(new JWSHeader(JWSAlgorithm.HS256), claimsSet);

 try
 {
    signedJWT.sign(signer);
 }
 catch (JOSEException e)
 {
    System.out.println("e2 : " + e);
 }

 // Serialize to compact form, produces something like
 // eyJhbGciOiJIUzI1NiJ9.SGVsbG8sIHdvcmxkIQ.onO9Ihudz3WkiauDO2Uhyuz0Y18UASXlSc1eS0NkWyA
 String s = signedJWT.serialize();
```

Figura 40: Generación de token

Se generará un token cifrado con una clave aleatoria de 32 bits (usando el algoritmo HS256) y una duración de una hora. Pasado ese tiempo el token dejará de tener validez y habrá que volver a identificarse.

Cuando el usuario quiera realizar una operación y nos envíe el token deberemos descifrarlo:

```
 // Check the token its correct
 SignedJWT signedJWT=null;
try
{
    signedJWT = SignedJWT.parse(token);
}
catch (ParseException e)
{
    e.printStackTrace();
}

 JWSVerifier verifier=null;
try
{
    verifier = new MACVerifier(StartServer.sharedSecret);
}
catch (JOSEException e)
{
    e.printStackTrace();
}

 try
 {
    if(signedJWT.verify(verifier))
     {
         if(signedJWT.getJWTClaimsSet().getExpirationTime().after(new Date()))
         {
             String Usuario = signedJWT.getJWTClaimsSet().getSubject();
             String Password = signedJWT.getJWTClaimsSet().getClaim("password").toString();
             return Usuario + ";" + Password;
         }
```

Figura 41: Descifrado de token

Habrá una fase de verificación que comprobará la fecha, así como la clave usada en el token, y nos devolverá el usuario y password correspondiente.

4.5.2.2 – Búsqueda en el directorio

Esta será la función principal del directorio ya que será la función más utilizada.

La búsqueda se realiza en base a filtros disponibles en la librearía unboundid usando la función *filter*:

En función del tipo de entrada que estemos buscando el filtro a crear será distinto. A la función filter.create le pasamos el string que queremos que contenga la entrada que buscamos. El asterisco al final indica que no nos importa lo que haya detrás, queremos todas las entradas que empiecen con ese string.

```
filter = Filter.create("cn=" + dato + "*");
```

Figura 42: Creación de un filtro sencillo

Podemos restringir todavía más la búsqueda usando varias condiciones por las que filtrar:

```
filter = Filter.create("cn=" + dato + "*");
filter2 = Filter.createExtensibleMatchFilter("objectClass",null,true, "groupOfNames");
filter=Filter.createANDFilter(filter,filter2);
```

Figura 43: Creación de un filtro más complejo

En este ejemplo estamos fijando que las entradas devueltas deben cumplir dos condiciones (que contengan el string cn = dato y que esa entrada contenga el atributo objectClass = groupOfNames). La relación entre los dos filtros la indicamos en la siguiente línea que nos creará un filtro AND con los dos filtros creados anteriormente.

Existen distintos filtros en función de la condición lógica que queramos aplicar (AND, OR, EQUAL,...)

Otra particularidad que vemos en el ejemplo es la función *createExtensibleMatchFilter.* Este filtro es especial ya que nos permite buscar entradas cuya información estructural (los atributos que definen de que clases hereda entre otros) coincida con los argumentos que le pasamos .Esta información estructural no está disponible de forma normal por lo que debemos usar está función especial.

La necesidad de combinar varios filtros surge por el hecho de restringir los resultados solo a lo que hemos buscado. En este caso estamos buscando entradas que tengan un atributo cn que coincida con el valor que buscamos. El problema es que el atributo cn le tienen varios tipos de entrada por lo que, si estamos buscando asignaturas, igual nos devuelve aulas y asignaturas ya que ambas entradas contienen ese atributo. Por ello debemos crear un filtro que cubra más condiciones que nos permita discernir entre estas entradas. En este caso por ejemplo, le pedimos que nos devuelva solo aquellas entradas que hereden la clase groupOfNames con lo cual solo nos devolverá las asignaturas.

Una vez definido el filtro a aplicar, realizamos la búsqueda usando la función *SearchRequest*:

```
SearchRequest busqueda = new SearchRequest(baseDN,SearchScope.SUB,filter);
try
{
   searchResult = conexion.search(busqueda);
}
```

Figura 44: Petición de búsqueda

Esta función necesita tres argumentos: la entrada de punto de partida de la búsqueda (puede ser la raíz del directorio o cualquier entrada que especifiquemos), el ámbito de búsqueda (hacia donde buscamos a partir del punto de partida) y el filtro a aplicar. En nuestro caso el punto de partida será la raíz del directorio y buscaremos entre todas las entradas que estén por debajo. De esta forma la búsqueda abarca todo el directorio.

El resultado de la búsqueda se almacena en la variable searchResult que luego habrá que recorrer para sacar los resultados a devolver.

4.5.2.3 – Creación de entradas

Este será el método de entrada de información del directorio. Una entrada en un directorio LDAP es un conjunto de atributos heredados de unas clases definidas.

Para definir los atributos que tendrá la entrada que queramos crear usaremos un array de atributos:

```
clases = new ArrayList<Attribute>();
```

Figura 45: ArrayList

Según el tipo de entrada que queramos crear tendremos que decidir que clases y atributos añadiremos. Vamos a verlo en el caso de querer crear un aula:

```
clase = new Attribute("objectClass","top","inetOrgPerson");
clases.add(clase);
```

Figura 46: Añadir atributos a una entrada (1)

Como vemos, creamos un nuevo atributo que serán las clases de las que heredará esa entrada y añadimos ese atributo al array. Lo siguiente será crear y añadir al array los atributos que contendrán la información.

```
clase = new Attribute("cn",input.Aula);
clases.add(clase);
```

Figura 47: Añadir atributos a una entrada (2)

Cuando ya tenemos todos los atributos definidos, debemos crear el DN de la entrada. Para ello usaremos los argumentos de entrada de la petición REST:

```
EntryDN = "cn=" + input.Aula + ",ou=" + input.Edificio + ",ou=edificios,dc=mydomain,dc=com";
```

Figura 48: Recreación del DN de la entrada

Ahora que ya tenemos toda la información necesaria para crear la entrada, pasaremos la petición de crear la entrada a la función *AddRequest* de la librería unboundid que tomará como argumentos el DN de la entrada nueva y el array de atributos.

```
entrada = new AddRequest(EntryDN,clases);
```

Figura 49: Petición de creación de una nueva entrada

Para terminar, enviamos la orden al directorio:

```
conexion.add(entrada);
```

Figura 50: Envío de la orden al directorio

Dado que esta es una función que nos permitirá gestionar el directorio, solo el usuario Manager tendrá permiso para crear entradas nuevas.

En la creación de entradas tenemos dos casos especiales de entradas. El primer caso será el de las asignaturas.

Al crear una nueva asignatura y para que sean efectivos los permisos definidos en el directorio, debemos modificar los ACL para incluir una línea nueva con la nueva asignatura. Para ello debemos modificar el archivo *slapd.conf* y después, reiniciar el directorio para que se hagan efectivos los nuevos permisos. Usaremos las funciones *FileReader, BufferedReader*, *PrintWritter* y *FileWriter* de la librería unboundid.

```
f = new FileReader (archivo);
br = new BufferedReader(f);
w = new FileWriter (nuevoArchivo);
bw = new PrintWriter (w);
```

Figura 51: Funciones de lectura y escritura de archivos

Como este archivo contiene toda la configuración del directorio, buscaremos una frase en concreto que pusimos en el fichero durante la instalación del directorio para indicar el inicio de la sección de los ACLs:

```
if(cadena.trim().compareTo("# Groups policy (ACls)")==0)
```

Figura 52: Cadena de comienzo de las ACL

Cuando encontremos esa frase, ya sabemos que podemos empezar a escribir las nuevas líneas. Una vez terminamos de modificar el archivo, debemos reiniciar el servicio slapd:

```
String cmd = "service slpad restart";
Runtime.getRuntime().exec(cmd);
```

Figura 53: Reinicio del servidor LDAP

Otra particularidad de las entradas de asignaturas, es que debemos decirle al menos un usuario que sea miembro de la asignatura. Como es posible que a la hora de crear la asignatura no se sepan quienes son los alumnos, introduciremos el usuario Manager como miembro de la asignatura, ya que este usuario, como tiene permiso sobre todo el directorio, no interferirá con los permisos propios de cada asignatura. La única pega será que debemos filtrar en los resultados de los usuarios para que este usuario no salga reflejado cuando nos pidan la lista de alumnos.

El segundo caso de entrada especial será el caso de los estudios. Si nos fijamos en la estructura del directorio vemos que hay dos nodos, que son alumnos y asignaturas, cuya única función es la de aportar organización y claridad al directorio, solo cumplen una función estructural, no contienen información. Por tanto, cuando creemos una nueva entrada de estudios, debemos crear estos dos nodos.

4.5.2.4 – Borrado de entradas

Esta función, siendo también de gestión, solo estará disponible para el usuario Manager. Para borrar una entrada, solo necesitamos el DN de esa entrada como argumento para la función *DeleteRequest* de la librería unboundid:

```
String EntryDN2= "ou=asignaturas,ou=" + input.Estudios + ",ou=estudios,dc=mydomain,dc=com";
entrada = new DeleteRequest(EntryDN2);
```

Figura 54: Función Delete

Luego le pasamos la orden al directorio con la función *delete:*

```
conexion.delete(entrada);
```

Figura 55: Enviar orden de borrado al directorio

Esta función tiene una particularidad, no podrás borrar una entrada si esa entrada tiene hijos (tiene entradas por debajo de esta).Esto es una limitación de la librería *unboundid*, aunque esta librería tiene otra función que si lo permite, pero solo está disponible en la versión de pago de la librería. Si bien es cierto que se podría haber implementado esta función recorriendo el árbol del directorio y borrando todos los hijos de la entrada a borrar, este proceso no sería igual de óptimo que la opción nativa de la librería, aparte que entraña el riesgo de que el usuario no sepa bien que es lo que está borrando y se cargue la estructura del directorio. De todas formas, en el uso normal, las entradas que se borrarán no tendrán hijos, por lo que no tendremos problemas.

En función del tipo de entrada que borremos debemos hacer ciertas operaciones después. Si borramos una asignatura, debemos borrar su línea correspondiente de las directrices de permiso (ACL).En un principio esto no es estrictamente necesario ya que al no existir la entrada sobre la que se aplica, no habría problemas pero dado el volumen de asignaturas que existirán en el directorio, si no borramos esas líneas el archivo slapd.conf tendría un tamaño desmesurado y estaría lleno de líneas que no tiene efecto.

Si la entrada que borramos es una persona debemos buscar en el directorio todas las asignaturas en las que esta persona estaba inscrita y borrar su registro.

El último caso particular es aquel en el que borramos una entrada de tipo *estudios.* En este caso, como ya explicamos en la parte de crear entradas, debemos borrar los nodos estructurales de *alumnos* y *asignaturas* antes de borrar la entrada de *estudios*.

4.5.2.5 – Modificación de entradas

Una vez creadas las entradas se puede dar el caso de que uno de los datos cambie por lo que debemos dar la posibilidad de cambiar solo un dato concreto de una entrada. Esto podremos hacerlo usando la función *Modification* de la librería unboundid:

```
switch(input.tipoModificacion)
{
    case "ADD":

        modificacion = new Modification(ModificationType.ADD,datoCambiar,input.valor);
        break;

    case "REPLACE":

        modificacion = new Modification(ModificationType.REPLACE,datoCambiar,input.valor);
        break;

    case "DELETE":

        modificacion = new Modification(ModificationType.DELETE,datoCambiar);
        break;
```

Figura 56: Tipos de modificación

A esta función debemos pasarle como argumentos el tipo de modificación, el nombre del atributo a cambiar y el nuevo valor a escribir. Como vemos en la captura de pantalla, hay tres tipos de modificaciones posibles, estas son las descripciones de la librería:

- ADD → añade el valor al atributo indicado. Si ya existe un valor en ese atributo, el nuevo valor se añadirá junto al valor ya existente.
- REPLACE → reemplaza el valor del atributo por el nuevo valor. Si no se indica un valor, se borrará el atributo entero. Si no existía un valor en el atributo, se añadirá el nuevo valor.
- DELETE → borra el valor del atributo. Si no se indica valor, borra el atributo entero.

En principio puede parecer que no hay problema con este modo de funcionamiento pero si nos fijamos en el funcionamiento de ADD, nos puede crear un problema con los atributos multi-valor (multi-value). Si no lo controlamos se puede dar el caso de un atributo que por su significado debe tener un solo valor (por ejemplo el DNI) y que mediante una modificación, acabemos teniendo más de un DNI en una entrada. Esto aparte de provocar que tengamos datos incorrectos puede provocar bugs. Para evitar esta situación debemos establecer unas comprobaciones en función del atributo y el tipo de modificación que se quiera realizar. En el caso de ADD comprobaremos si existe un valor previo en ese atributo, si existe, devolveremos un error con un mensaje diciendo que se use la modificación REPLACE.

Estas comprobaciones las extenderemos también a los demás tipos de modificación ya que si nos borran un atributo que consideramos obligatorio, y luego vamos a buscarlo, nos puede producir un comportamiento inesperado.

Una vez resuelto esto también debemos traducir el nombre del atributo entre el que nosotros le hemos dado y el que le corresponde en el directorio (web=labeledURI, Nombre=ou, etc).Como último paso, debemos reconstruir el DN de la entrada a modificar.

Con todos los argumentos creamos la petición de modificación y se la pasamos al directorio:

```
ModifyRequest peticion = new ModifyRequest(EntryDN,modificacion);
conexion.modify(peticion);
```

Figura 57: Petición de modificación y envío al directorio

Una vez modificado el valor del atributo y depende de que atributo sea, en algunos casos debemos buscar el antiguo atributo en el directorio y sustituirlo por el nuevo en todas las entradas en las que estuviera presente.

Como en otras funciones tenemos casos especiales en función del atributo que queramos modificar. La mayoría de atributos en el directorio son multi-value aunque nosotros les restrinjamos a que tengan un solo valor, pero hay algunos que necesitamos que sigan siendo multi-value, como por ejemplo, los alumnos de una asignatura. En este caso las restricciones que hemos hecho antes a los tipos de modificación habrá que adaptarlas ya que debemos poder añadir nuevos valores aunque ya existan valores en ese atributo, y en el caso de reemplazar un valor por otro, debemos indicar cuál es el valor anterior.

Otro de los casos especiales es el de cambiar contraseña, ya que aparte de aportar un nuevo valor, debemos cifrarlo. Para ello usaremos la función *getPasswordLDAP* que nos cambia el valor y lo cifra directamente.

```
boolean resultado = getPasswordLDAP(EntryDN,input.valor,conexion);
```

Figura 58: Generar password

El último caso especial se produce cuando queremos cambiar el nombre de la entrada (el DN).En este caso no nos sirve la modificación normal, debemos usar la función *ModifyDNRequest:*

```
ModifyDNRequest modifyDNRequest = new ModifyDNRequest(EntryDN, datoCambiar + "=" + input.valor, true);
```

Figura 59: Función ModifyDN

En este caso y debido a usar la versión gratuita de la librería, no se nos permite cambiar el nombre de una entrada que tenga hijos.

En el caso de cambiar el nombre de una asignatura o un plan de estudios (forma parte del DN de una asignatura) debemos modificar también el contenido de las ACL en el archivo slapd.conf.

Para terminar, como función especial, si modificamos el atributo “estudiantes” de una asignatura, y ponemos como valor “All”, borraremos todos los estudiantes asignados a esa asignatura. Esto se hace para a la hora de cambiar de curso (que toca renovar la lista de estudiantes) poder borrarlos todos en una sola operación para luego importar la nueva lista de alumnos desde un archivo.

4.5.2.6 – Importar alumnos para una asignatura

Teniendo en cuenta que una vez al año cambian los alumnos de una asignatura, al menos una vez al año debemos cambiar los miembros de la entrada. Ya hemos solucionado el problema de borrarlos todos y ahora tenemos el problema de introducir los nuevos alumnos. Ya tenemos una opción disponible para ello con la opción de

modificar entradas. Ahora nos ponemos en el caso y vemos que tendremos que hacer en los peores casos, más de cien llamadas a esa función, una por alumno. Si bien esto es posible, no es para nada cómodo ni rápido. Asique lo mejor es implementar una función para meter todos los alumnos a la vez. Los alumnos a incluir en la asignatura se cogerán de un archivo de texto plano con el siguiente formato:

"Nombre, Apellido1 Apellido2"

Usaremos las funciones *FileReader* y *BufferedReader* para leer los nombres del archivo y para cada nombre usaremos la función *ModifyRequest* para pasarle la petición de modificación del atributo de la entrada al directorio:

```
Modification modificacion = new Modification(ModificationType.ADD,"member",MemberDN);
ModifyRequest peticion = new ModifyRequest(EntryDN,modificacion);
```

Figura 60: Función de modificación y envío al directorio

Para poder añadir a un alumno a la asignatura, la entrada correspondiente del alumno debe existir. Si se produce algún error o el alumno no existe, se pasará al siguiente nombre del fichero hasta terminar de leerle, y al final de la operación se indicará cuantos alumnos no se han podido importar.

De esta forma en una sola operación metemos a todos los alumnos nuevos en una asignatura.

4.5.2.7 – Exportar información del directorio

Llegado a este punto, nos planteamos que puede ser interesante, el sacar información del directorio a un archivo de texto plano. Esta función nos hará las veces de backup de los datos del directorio.

Para ello usaremos la función de búsqueda que ya hemos desarrollado para buscar información en el directorio y estableceremos el punto de partida de la búsqueda en el elemento que nos indiquen. De esta forma cogeremos toda la información de todas las entradas que estén por debajo del punto de partida. Si escogemos como punto de partida el elemento raíz del directorio, el resultado será un backup de toda la información contenida en el directorio.

Luego usaremos las funciones que ya hemos visto para escribir en un archivo la información deseada.

El aspecto del archivo de información será este:

```
dn: ou=profesores,dc=mydomain,dc=
objectClass: top
objectClass: organizationalUnit
ou: profesores

dn: uid=profesor1,ou=profesores,d
cn: profesor1
uid: profesor1
objectClass: top
objectClass: posixAccount
```

Figura 61: Archivo exportación

Como vemos la información aparece tal cual se almacena en el directorio. Esto se deja así y no se "traduce" por una cuestión de rendimiento. Si tenemos que "traducir" toda la información haría que requiriese más tiempo y recursos la operación, y en caso de querer usar ese backup para restaurar la información del directorio, tendríamos que volver a "traducirlo" lo que nos llevaría más tiempo y recursos todavía.

Tal cual está desarrollada esta función nos puede servir para obtener la información de una sola entrada (en el caso de que no tenga hijos), una rama del árbol, o de todo el directorio. Como es una función de gestión del directorio, solo el usuario Manager estará capacitado para usarla.

4.5.2.8 – Importar información al directorio

Una vez tenemos la información del directorio en un archivo de texto plano, tenemos que habilitar la opción de incorporar esa información al directorio para terminar de desarrollar la función de backup. Como ya hemos visto en otros apartados, usaremos las funciones para leer de un archivo de texto la información de las entradas y las añadiremos al directorio. Dado que es una operación de gestión solo el usuario Manager tendrá permiso para ejecutarla.

Si al importar la información, alguna entrada ya existiese, se daría un fallo. Lo normal sería borrarlo pero nos podemos topar con el caso de que esta entrada tenga hijos y no nos permita borrarla. Por lo tanto, antes de importar la información, habría que borrar la información previa.

4.5.2.9 – Entradas

En las funciones de las entradas lo único que se hará será coger la información contenida en la entrada y devolverla. Si nos fijamos, toda la información que se devuelve no está contenida dentro de la entrada. Parte de la información que devolvemos está contenida en la propia estructura de árbol del directorio. Por ejemplo, si queremos saber que asignaturas forman parte de un plan de estudios, solo tenemos que mirar los hijos de la entrada correspondiente al plan de estudios. Lo mismo podemos decir de las aulas de un edificio. Habrá otras informaciones que no serán tan fáciles de obtener porque están en entradas no directamente relacionadas, como por ejemplo las asignaturas en las que está inscrito un alumno. Para ello buscaremos las asignaturas en las que el alumno sea miembro. Una vez obtenida toda la información que queremos devolver, lo formateamos como json y lo devolvemos.

4.5.3 – Instalación del API

Cuando ya hayamos terminado el desarrollo del API debemos instalarlo en un servidor para poder hacerle peticiones. El primer paso es exportar el proyecto. Para ello, desde el eclipse, vamos a File → Export → Web → WAR file

Con el archivo WAR ya generado, nos queda instalarlo en el servidor. Para ello debemos para el servicio tomcat, copiar el archivo en la carpeta /opt/tomcat/webapps/ y después reiniciar el servicio tomcat.

```
[root@server ~]#cp LdapServlet.war /opt/tomcat/webapps/LdapServlet.war
```

Figura 62: Copiar servlet a la ruta de Tomcat

4.5.4 – Verificación del servicio

A medida que vamos implementando funciones del servicio debemos ir probando su funcionamiento. Para ello emplearemos el programa SoapUI que nos permitirá realizar peticiones al API y ver los resultados así como buscar posibles fallos. Las pruebas realizadas para este proyecto se pueden encontrar en el fichero REST-Project-1-soapui-project.xml.

El funcionamiento del programa es sencillo, indicamos los datos que queremos mandar al servidor, le indicamos la IP y el servidor nos devolverá los resultados. Se plantean diferentes escenarios intentando forzar el API para descubrir posibles bugs y arreglarlos, simulando el uso que puede hacer después un usuario del API.

Gracias a este proceso se han resuelto diversos bugs relacionados sobre todo con la ausencia de elementos necesarios en las peticiones.

4.5.5 – Seguridad

Para terminar el desarrollo del API, debemos asegurar el acceso al servidor que lo contiene con un cierto nivel de seguridad. Para ello haremos uso del protocolo HTTPS, por lo que tendremos que crear un certificado y configurar el servidor que contendrá el API.

Para crear el certificado usaremos la herramienta *keytool* que está incluida en el sdk de java:

```
[root@MiWiFi-R1CM-srv Ruben]# keytool -genkey -alias LDAPCentos -keypass CentosLDAP -keystore /opt/tomcat/webapps/localhost.b
in -keyalg RSA
Enter keystore password:
What is your first and last name?
  [Unknown]:
What is the name of your organizational unit?
  [Unknown]:
What is the name of your organization?
  [Unknown]:
What is the name of your City or Locality?
  [Unknown]:
What is the name of your State or Province?
  [Unknown]:
What is the two-letter country code for this unit?
  [Unknown]:
Is CN=Unknown, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown correct?
  [no]:  Y
```

Figura 63: Creación del certificado mediante la herramienta keytool

Con el certificado ya creado, debemos habilitar el servidor para que acepte conexiones SSL. Para ello debemos editar el archivo server.xml en la carpeta /opt/tomcat/conf/.

Debemos descomentar la sección referente a las conexiones SSL y configurarla para usar el certificado que hemos creado:

```
<Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
           maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
            enableLookups="true" disableUploadTimeout="true" acceptCount="100"
            SSLEnabled="true" scheme="https" secure="true"
           clientAuth="false" sslProtocol="TLS"
            keystoreFile="/opt/tomcat/webapps/localhost.bin"
            keystorePass="CentosLDAP" keyAlias="Centos" />
```

Figura 64: Configuración conexión SSL

Ahora el servidor tomcat ya acepta conexiones SSL que usen el certificado que hemos creado conectándose al puerto 8443.

Como el certificado es uno que hemos generado nosotros, al acceder desde un navegador nos saldrá una advertencia ya que no reconoce el certificado, al no ser proveniente de una entidad certificadora. Sin embargo para el ámbito de este proyecto, cumple su cometido.

4.6 – Página Web

Una vez desarrollado el API, la mejor forma de asegurar de que cumple su cometido, es decir que sirve para desarrollar aplicaciones, es desarrollar una aplicación que haga uso de ella.

Haremos una página web que nos permitirá hacer uso de todas las funciones desarrolladas en el API con lo que podremos consultar y gestionar el directorio LDAP desde cualquier navegador.

El primer paso será montar un servidor web que contendrá nuestra página web. Para ello usaremos otra máquina virtual centOS en la que instalaremos HTTP Apache:

```
[root@server ~]# yum install httpd
```

Figura 65: Instalar servidor HTTP Apache

Una vez instalado, lo iniciamos y lo automatizamos para que arranque al inicio del sistema:

```
[root@server ~]# service httpd start
```

Figura 66: Iniciar el servicio del servidor HTTP

```
[root@server ~]# chkconfig httpd on
```

Figura 67: automatizar el inicio del servidor HTTP

Los archivos que conformen la página web se guardarán en /var/www/html.

Una vez puesto en marcha el servidor web, empezaremos a desarrollar la página web. Dado que tanto como la web como la app Android son pruebas de uso de la API, se le dará un estilo muy básico, por lo que haremos todo el desarrollo directamente con un editor de textos, como el gedit. Para el estilo de la web haremos un pequeño archivo css.

El funcionamiento de la página será sencillo, creamos unos formularios donde el usuario introducirá la información, creamos un json con esos datos que enviaremos a la API que nos devolverá otro json con la respuesta, que parsearemos y mostraremos en pantalla.

Dado que no sabemos la información que vamos a tener que mostrar en la página web, usaremos PHP que nos permitirá generar código HTML de forma dinámica en función de la información que tengamos. PHP no viene por defecto en el sistema operativo por lo que tendremos que instalarlo:

```
[root@MiWiFi-R1CM-srv Ruben]# yum install php
```

Figura 68: Instalación PHP

Con esto ya tenemos instalado PHP y cURL.

```
<FORM name="loginForm" ACTION="https://192.168.31.153:443/login.php"  method="post">
<div id=bloque1>
<div id=mensaje>
<p>Bienvenido al servicio de directorio LDAP de la Universidad de Cantabria</p>
</div>


<div id=cadena1>
<p>Usuario : <input type="text"  name="user" value="usuario"></p>
</div>
<div id=modificar>
<p>Password : <input type="password" name="password" value="password"></p>
</div>
<div id=acceder>
<input type="submit" value="Acceder">
</div>
</FORM>
```

Figura 69: Formulario de la pantalla de login

```
 // cojo los valores de usuario y password del formulario
 $Usuario = $_POST['user'];
 $Password = $_POST['password'];
 $nuevo = $_POST['newUser'];

// creo el json para enviar al servicio
//creo una clase login
class Login
{
   var $Usuario;
   var $Password;

   function __construct($Usuario,$Password)
   {
      $this->Usuario = $Usuario;
      $this->Password = $Password;
   }
}
$login = new Login($Usuario,$Password); // instancio la clase
$json = json_encode($login);// creo e imprimo el json
```

Figura 70: Creación del json a enviar al API

Para poder comunicarlos con el API usaremos PHP y en concreto, la librería cURL que nos permitirá hacer conexiones HTTP POST usando SSL:

```
// Comunicamos con el servicio rest usando curl
$ch = curl_init('https://192.168.31.248:8443/LDAPserver/rest/');
curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "POST");
curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, true);
curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, true);
curl_setopt($ch, CURLOPT_CAINFO, "/home/Ruben/Desktop/WebPage/as.pem");
curl_setopt($ch, CURLOPT_POSTFIELDS, $json);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
curl_setopt($ch, CURLOPT_HTTPHEADER, array(
    'Content-Type: application/json',
    'Content-Length: ' . strlen($json))
);
$result = curl_exec($ch);
```

Figura 71: Comunicación con el servicio web usando cURL

En la captura podemos ver cómo le indicamos a cURL la URL a la que se debe conectar, el método a usar (POST), la ruta del certificado y el tipo de información que vamos a enviar (json) y se quedará a la espera de la respuesta del API.

```
// decodificamos la repuesta
$token=json_decode($result);
```

Figura 72: Decodificación de la respuesta

Como vemos este es el ejemplo más sencillo porque la única respuesta del API es el token, por lo que solo hay un dato que parsear. En función de la respuesta decidiremos que página cargar o que mensaje mostrar.

```
if(strlen($token->token)!=0)
{
   session_start();
   $_SESSION['user']=$Usuario;
   $_SESSION['token']=$token->token;
   $_SESSION['anonymous']=1;
   if($nuevo=="nuevo")
   {
       include("newUser.php");

   }
   else
   {
       include("welcome.php");
   }
}
else
{
   echo '<p>Usuario o password incorrectos</p>';
}
```

Figura 73: Flujo de la web en función de la respuesta

En función de si me responde con un token, lo cual significaría que el login ha sido correcto, decido si ir a la siguiente página o devolver un error. Esto se podría hacer también en función de los códigos HTTP que devuelva en la cabecera el API. Como hemos dicho antes, está página es una prueba y demostración del API por lo que habrá mejores formas de desarrollarla.

Como vemos también en las capturas usaremos sentencias PHP para determinar el flujo de la página web en función de la información devuelta por el API. PHP también nos servirá para resolver el problema de las variables, ya que en HTML no podemos guardar variables entre una página y otra de forma sencilla, por lo que usaremos variables de sesión que nos permite PHP para guardar variables de una página a otra, como el token.

El esquema que hemos visto en este ejemplo será el mismo para todas las páginas que formarán nuestra web, la única diferencia será que tendremos más o menos elementos en pantalla, por lo que no entraremos a explicar los pormenores de cada página.

En algunos casos, debemos hacer más de una petición al API para saber las entradas que existen de un tipo, como por ejemplo, al crear una asignatura, debemos indicar a que estudios pertenece. Podemos dejar que el usuario introduzca el nombre de los estudios, o hacemos una consulta al API para que nos diga todos los estudios ya existentes y los desplegamos en una lista. De esta forma nos aseguramos de que los estudios ya existen y nos evitamos errores al introducir los datos por parte del usuario.

Otra de las particularidades que nos pueden servir de ayuda es saber que si yo pido los datos de mi usuario, el API me devolverá mi DNI, si es cualquier otro usuario(a excepción del Manager) nos devolverá ese campo vacío. Podemos aprovechar la presencia de este campo para saber que somos nosotros los que estamos viendo nuestros datos, y en este caso, mostrar la posibilidad de modificarlos.

También debemos tener en cuenta el usuario que está navegando por la web ya que si es el Manager deberemos mostrar las opciones de gestión del directorio. En caso de que no lo sea se mostrarán otras pantallas distintas.

4.6.1 – Seguridad en la Web

Hasta ahora hemos implementado la seguridad en el API, pero ahora se nos plantea otro problema. En la página web, el usuario no accede directamente al API, si no que accede a la página y ésta accede al API. Por lo tanto si queremos que la conexión sea completamente segura, debemos crear otro certificado para la conexión entre el usuario y la web.

Para garantizar la seguridad de la conexión con la web necesitamos tres archivos:

- SSLCertificateChainFile → certificado CA intermedio. Indica los diferentes certificados desde nuestra página hasta la entidad certificadora final.
- SSLCertificateFile → certificado de la entidad certificadora final
- SSLCertificateKeyFile → clave usada en el certificado

En este caso, al igual que con la API, no tenemos entidad certificadora, por lo que firmaremos nuestro certificado nosotros mismos.

Debemos crear un CSR (Certificate Signing Request) y una llave privada usando OpenSSL.

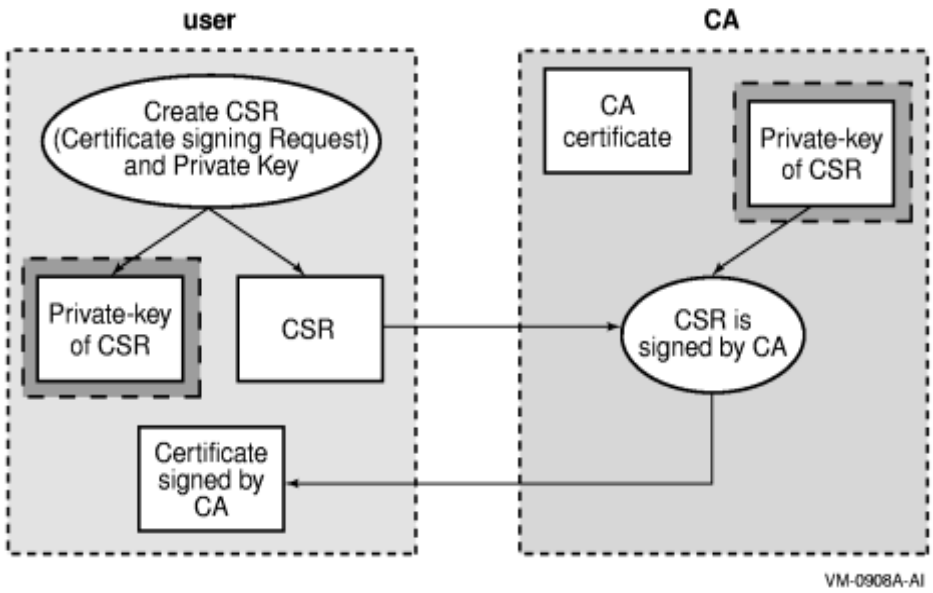

Figura 74: Esquema de certificados en la web

Para crear la llave privada haremos lo siguiente:

```
openssl genrsa -des3 -out www.mydomain.com.key 2048
```

Figura 75: Generación de la llave privada

Nos pedirá una frase que será nuestra clave y nos generará nuestra llave privada RSA 2048.

El siguiente paso será generar nuestro CSR:

```
openssl req -new -key www.mydomain.com.key -out www.mydomain.com.csr
```

Figura 76: Generación del CSR

Nos pedirá nuestra frase, que deberá ser la misma que la de nuestra llave privada. Después nos pedirá la información necesaria para el certificado (localidad, ciudad, país, etc) y nos devolverá el CSR creado.

Como certificado intermedio descargaremos uno genérico de Mozilla. Al tratarse de uno genérico no servirá como parte de la seguridad, ya que al ser autofirmado no hay ningún nodo entre la página web y la entidad certificadora(es el mismo equipo) pero puede dar problemas en algunos equipos si no existe un CA intermedio. Esta parte de seguridad no es del todo correcta ya que estamos usando un certificado autofirmado, por lo que no tenemos una entidad certificadora detrás que asegure que nuestro certificado es correcto.

En la implementación final deberemos comprar un certificado a una entidad registrada lo que nos asegurará la conexión.

Una vez creados los ficheros, los metemos en la carpeta /etc/pki/tls/ y configuramos el servidor para que los use modificando el archivo ssl.conf ubicado en la carpeta /etc/httpd/conf.d:

```
#   Server Certificate:
# Point SSLCertificateFile at a PEM encoded certificate.  If
# the certificate is encrypted, then you will be prompted for a
# pass phrase.  Note that a kill -HUP will prompt again.  A new
# certificate can be generated using the genkey(1) command.
SSLCertificateFile /etc/pki/tls/certs/www.mydomain.com.crt

#   Server Private Key:
#   If the key is not combined with the certificate, use this
#   directive to point at the key file.  Keep in mind that if
#   you've both a RSA and a DSA private key you can configure
#   both in parallel (to also allow the use of DSA ciphers, etc.)
SSLCertificateKeyFile /etc/pki/tls/private/www.mydomain.com.key

#   Server Certificate Chain:
#   Point SSLCertificateChainFile at a file containing the
#   concatenation of PEM encoded CA certificates which form the
#   certificate chain for the server certificate. Alternatively
#   the referenced file can be the same as SSLCertificateFile
#   when the CA certificates are directly appended to the server
#   certificate for convinience.
#SSLCertificateChainFile /etc/pki/tls/certs/server-chain.crt
```

Figura 77: Configuración de certificados

Es posible que aparte de estas modificaciones haya que descomentar la línea que activa el SSL en el mismo archivo:

```
SSLEngine on
```

Figura 78: Activación de SSL

Después de modificar el archivo debemos reiniciar el servidor HTTP.

En este punto ya tenemos activado el SSL y podemos acceder a la versión segura de nuestra web, sin embargo, seguimos pudiendo acceder a la versión no segura, por lo que debemos desactivar esta opción. Para ello modificaremos el archivo httpd.conf (/etc/httpd/conf) y comentaremos la línea correspondiente a la escucha por el puerto 80.De esta forma una petición que nos llegue por el puerto 80 (una petición normal HTTP) no contestará mientras que si nos conectamos al puerto 443(una petición HTTPS) podremos acceder a la web.

Una vez creado esto, cuando nos conectemos a la web deberemos descargar el certificado e incluirlo en nuestra lista de certificados aceptados. Aun así nos saldrá una advertencia indicando que no nos podemos fiar del certificado, ya que al ser autofirmado, no puede ser verificado por una entidad certificadora.

4.7 – App Android

Por último haremos una app Android cuya única función será la de consultar datos. Se ha decidido hacer así dado que el tamaño de pantalla normal de los terminales no permite una interfaz adecuada para editar datos y realizar tareas de gestión y teniendo en cuenta que las funciones de consulta serán más usadas que las de gestión.

El público objetivo de esta app serán smartphones con un tamaño de pantalla de 5 pulgadas y resolución HD o FullHD. En principio la versión de Android para la que se desarrolla será Android 4.2 (API 17) por lo que será compatible con versiones superiores a esta. Se ha decidido usar esta versión para que se alcance el mayor número de terminales posibles en los que la app se ejecute correctamente.

Dado que la interfaz está diseñada para smartphones, es posible que si ejecutamos esta app en un tablet o un smartphone de mayor/menor diagonal o resolución, no se muestre correctamente. Comentar también que dado que esta app es una prueba de uso de la API y no es definitiva, no se implementarán en esta app las líneas de diseño de Android Material Design ya que entiendo que en la versión final habrá que adaptar el diseño siguiendo el estilo de las otras páginas y servicios de la universidad.

Si un usuario quiere realizar labores de gestión desde un terminal móvil siempre puede acceder a la página web desde el navegador del smartphone.

En un principio se había decidido utilizar APP Inventor para realizar la App de forma sencilla, pero al intentar realizar la primera conexión al API REST encontramos el problema de que APP Inventor no se maneja bien con las conexiones SSL. Si bien había alguna forma más o menos compleja de sortear este obstáculo, en este punto se decidió que era mejor desarrollar la app directamente en Java para evitar limitaciones. Para ello se usará la herramienta Android Studio.

El primer paso en Android Studio será desarrollar la interfaz que tendrá la app. En una app Android las interfaces se guardan en la carpeta res del proyecto en un archivo xml y luego en el código java se decide que interfaz cargar y que funciones ejecutar cuando se pulsa algún elemento de la interfaz. La estructura del proyecto de la app quedará así:

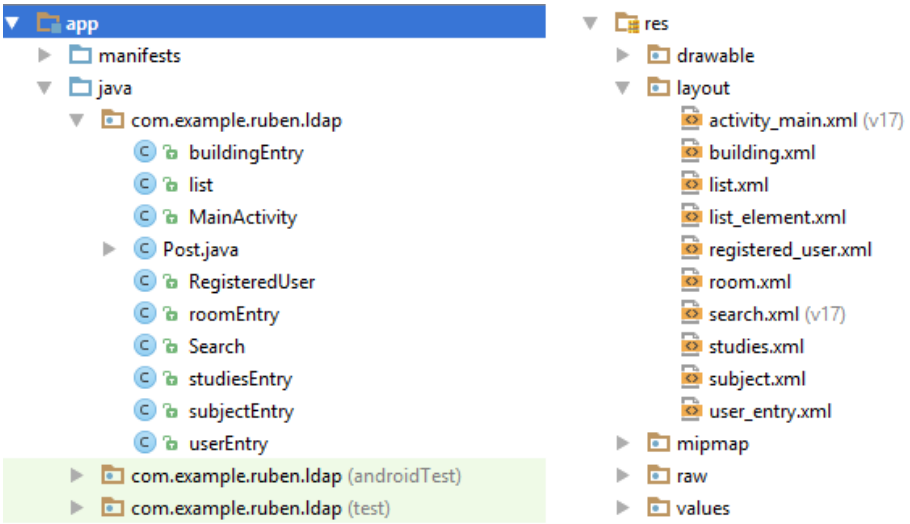

Figura 79: Desglose de recurso de la app

En la carpeta *res* se guardarán todos los recursos que tendrá que usar nuestra app. La carpeta *layout* contendrá las interfaces de las pantallas. En la carpeta *raw* está el certificado que debemos usar para conectarnos al API y en la carpeta *values* hay un archivo llamado *strings.xml* que contendrá todos los strings que usemos en nuestra app.

Android Studio proporciona una interfaz que nos permitirá crear nuestras interfaces fácilmente permitiéndonos cambiar las propiedades de un objeto de forma gráfica y viendo en tiempo real el cambio. Sin embargo si lo deseamos podemos editar directamente el xml que define nuestra interfaz.

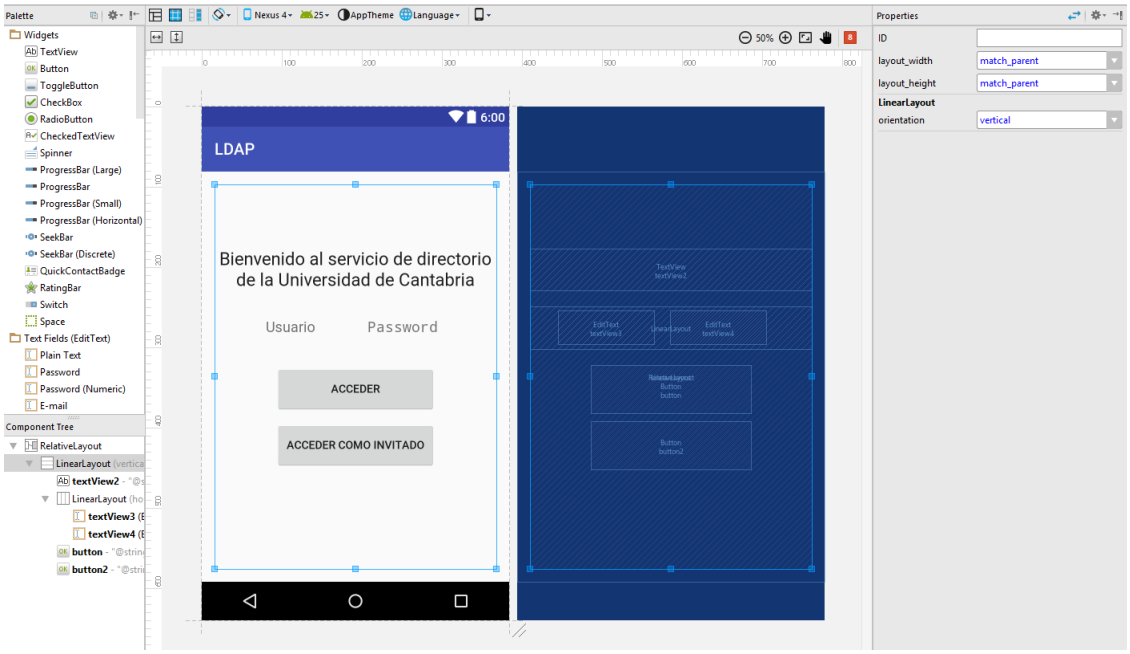

Figura 80: Interfaz AndroidStudio

Las pantallas creadas para esta app no son nada complejas, en la mayoría solo tenemos unos campos de texto que rellenaremos con los datos que nos devuelva el API, o unos que deberá rellenar el usuario como los de usuario y password que vemos en la imagen. La interfaz más compleja que hemos tenido que crear ha sido crear una lista que debía generarse automáticamente en función del número de elementos y con scroll (desplazamiento) vertical.

Una vez creada la interfaz debemos crear el código que se ejecutará cuando pulsemos sobre los elementos de la interfaz. Crearemos una clase por cada interfaz. Todas las clases tienen el método *onCreate* que indica lo que debemos hacer cuando se carga esa clase. Aquí le diremos la interfaz que debe cargar:

```
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
```

Figura 81: Método de carga de la interfaz

Todos los botones que creemos en las interfaces tienen el método *setOnClickListener* al que podemos asociar una función que se ejecutará cuando pulsemos ese botón:

```
// we set action for button acceder como invitado
 final Button acceder_guest = (Button) findViewById(R.id.button2);

acceder_guest.setOnClickListener((view) → {
        Intent intent = new Intent(MainActivity.this, Search.class);
        startActivity(intent);
        //finish();
});
```

Figura 82: Asignación de métodos a pulsaciones de botones

En este caso, cuando pulsemos sobre el botón "acceder como invitado" cargaremos la siguiente clase, que será la clase Search, cuyo método *onCreate* cargará la interfaz de búsqueda.

Tendremos que crear también una clase que no tendrá interfaz asociada, que será la clase POST y estará encargada de comunicarse con el API. Para ello al igual que en la web, debemos crear el json correspondiente, que le pasaremos a esta clase, junto con el certificado SSL y la URL a la que debemos enviarlo. Esta clase le enviará el JSON y nos devolverá el JSON de respuesta del API. Comentar que aquí, al contrario que en la web, solo debemos usar el certificado del API ya que nos conectamos directamente.

Una vez con el JSON que nos devuelva el API con el resultado de la operación procederemos a mostrar los resultados en pantalla si corresponde. En la mayoría de los casos será rellenar un cuadro de texto usando el método *setText:*

```
name.setText(result.getString("Name"));
```

Figura 83: Método para rellenar la interfaz con los datos

Como el app será solo de consulta de datos, no tendremos que hacer diferenciación entre los diferentes tipos de usuarios, para todos las pantallas serán las mismas.

Aprovechando que Android por defecto trae instaladas varias apps de Google como gmail, Google maps, etc hemos decidido integrar la app con estas aplicaciones. Por ejemplo en los campos que sabemos que son direcciones de email, hemos programado la app para que al pulsar sobre ello, abra gmail con la dirección de email sobre la que hemos pulsado como destinatario:

```
//lanzar gmail cuando pulsamos en email
final TextView textViewEmail = (TextView) findViewById(R.id.textView8);

textViewEmail.setOnClickListener((view) → {

        Intent mail=null;

        try
        {
            mail = new Intent(Intent.ACTION_SENDTO);
            email2="mailto:"+email2;
            mail.setData(Uri.parse(email2));
            startActivity(mail);
        }
        catch (Exception e)
        {
            System.out.println(e);
            Toast.makeText(userEntry.this, "Data not valid or no application found",Toast.LENGTH_LONG).show();
        }

});
```

Figura 84: Integración con aplicaciones de sistemas

Esto mismo hemos hecho con los enlaces a páginas web (que lanzarán el navegador), con los números de teléfono (que lanzarán el marcador) y con las coordenadas o direcciones (que abrirán Google maps).

Por defecto Android no nos dejará ejecutar cualquier clase en nuestra app. Cada clase java que definamos que queramos usar debemos declararla en nuestro manifiesto .El archivo *AndroidManifest.xml* contiene la información de las actividades que permitimos que se ejecuten dentro de nuestra app. Por ello cada vez que creemos una clase nueva, debemos declararla dentro de este fichero:

```
<activity android:name=".Search">
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />

        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

Figura 85: Modificación de AndroidManifest

Una vez tenemos el app más o menos terminada, toca el turno de probarla. Para ello tenemos varias opciones. Podemos crear el ejecutable e instalarla en nuestro teléfono, conectar el teléfono a nuestro pc y ejecutarla desde el pc o crear un dispositivo virtual. La primera opción no es recomendable ya que en caso de fallo, no tenemos información de que puede haber fallado. Las otras dos opciones son válidas, sin embargo nosotros hemos escogido la tercera por dos razones. La primera es que puedes simular un dispositivo que no poseas con diferentes resoluciones o tamaños de pantalla y la segunda es que a veces es difícil conectar el dispositivo para ejecutar aplicaciones no firmadas ya que son necesarios los drivers *adb* del dispositivo. Sin embargo cualquiera de estas dos opciones nos permite ver el log de ejecución de la app mientras la ejecutamos lo que nos permite saber el punto exacto donde falla.

En nuestro caso hemos decidido crear un dispositivo virtual, en este caso un Nexus 4 que cuenta con una pantalla HD (1280x720 pixeles). Para crear el dispositivo virtual debemos ir a run → run App → create new virtual device:

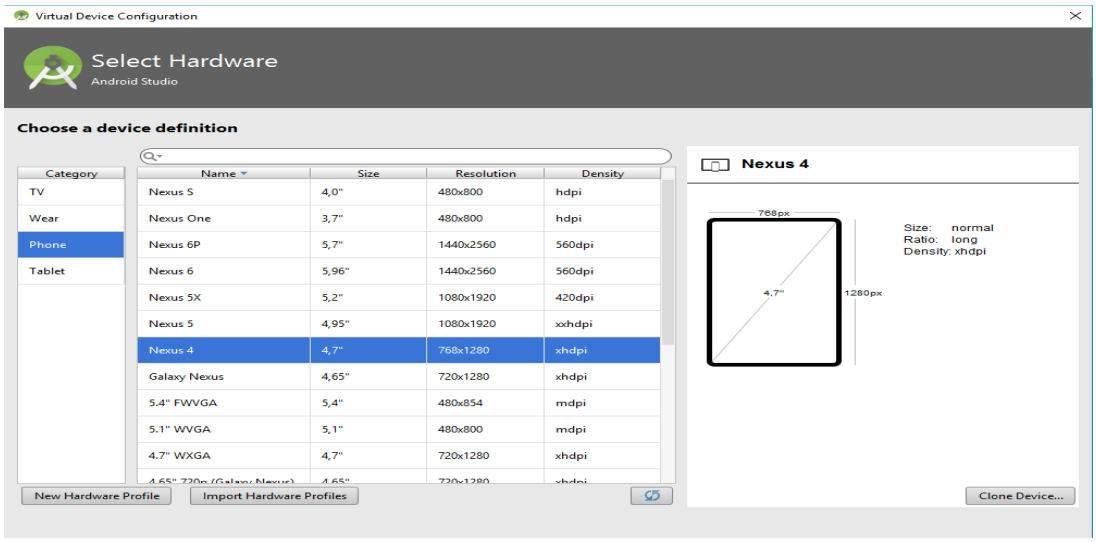

Figura 86: Interfaz creación dispositivo virtual

Nos pedirá seleccionar una imagen del sistema operativo Android lo cual nos permitirá simular también el funcionamiento de nuestra app en distintas versiones del sistema operativo. En nuestro caso hemos seleccionado la versión Android Marshmallow (6.0) ya que es una de las versiones más extendidas actualmente. Como ya hemos comentado antes la app debería funcionar sin problemas en versiones de Android superiores a la versión 4.2, siendo posible su funcionamiento en versiones anteriores aunque no es posible asegurarlo.

Es posible que nos pida instalar ciertos complementos y habilitar ciertas opciones de la BIOS para aumentar el rendimiento del simulador.

Una vez ejecutado podemos probar nuestra aplicación en el simulador y solucionar posibles errores.

Figura 87: Ejemplo de pantallas de la app

Cuando ya hayamos testeado lo suficiente nuestra app, procederemos a crear el ejecutable para instalarla. Para ello vamos a build → build apk.

Como es una app que no está firmada ni publicada en Google Play, el dispositivo en el que vayamos a instalarla nos advertirá de que no es una aplicación segura y no nos dejará instalarla. Para poder saltarnos esta restricción debemos ir a los ajustes de nuestro dispositivo, ir a la sección *seguridad* y activar la opción orígenes *desconocidos*. Ahora ya nos debería dejar instalar el apk y podríamos ejecutarla sin problemas. En este caso se ha probado el funcionamiento de la app en un dispositivo FULL HD de 5 pulgadas con Android Marshmallow.

5 – Conclusiones y mejoras

Llegados a este punto podemos decir que hemos cumplido los objetivos de este proyecto. Se ha desarrollado un API que actúa sobre un directorio LDAP y que no necesita conocimientos del protocolo. Con el API hemos demostrado que se pueden desarrollar aplicaciones de distinto índole y enfoque haciendo uso de todas las funciones que nos permite el API.

Hemos visto como LDAP es un complemento perfecto a las bases de datos actuales y, en ciertos escenarios, un sustituto de ellas en los casos que se trate de poca información , de información protegida o sensible ,y además se cumpla que sea información que no cambie con frecuencia (LDAP se especializa en la lectura de datos). LDAP aporta más seguridad que SQL, lo que nos permite despreocuparnos de los ataques propios de SQL (como SQL injection) en parte por no ser tan usado ni conocido. Además soporta conexiones SSL por lo que la conexión es totalmente segura. Por estas características es perfecto para almacenar datos de logueo de usuarios. Sin embargo tampoco está libre de ataques ya que tiene su equivalente a SQL (LDAP injection).

Una vez concluido el trabajo, hemos visto como el API que hemos desarrollado sirve en efecto para desarrollar aplicaciones sin tener ningún conocimiento de LDAP. Sin embargo esto es solo el comienzo del proyecto. Hay varias formas de mejorarlo, siendo el primer paso un testeo y uso por parte de personas ajenas al proyecto. De esta forma podemos obtener feedback de gente que no conoce los entresijos y nos puede ayudar a mejorar su uso y funcionamiento, así como añadir nuevas funciones que no se nos hayan ocurrido. Otra forma de ampliar el API sería prepararlo para aceptar otras representaciones como XML y que se pudiera acceder también por SOAP. En el ámbito de la Universidad de Cantabria se podría realizar una integración con los sistemas ya existentes, como por ejemplo usar el mismo sistema de logueo que ya se usa por ejemplo en el campus virtual. Esta integración conllevaría también rediseñar la página web y el app Android para adaptarlo al estilo visual de la universidad. Por último y por abarcar todos los ámbitos posibles, sería recomendable realizar una app para sistemas IOS.

6 – Referencias

[LDAP]

1. **http://www.zytrax.com/books/ldap/**

   Autor: ZyTrax,Inc., **Año publicación:** 2016, **Última visita:** Mayo 2016

2. **http://www.zytrax.com/books/ldap/ape/**

   Autor: ZyTrax,Inc., **Año publicación:** 2016, **Última visita:** Mayo 2016

3. **http://www.zytrax.com/books/ldap/apa/types.html**

   Autor: ZyTrax,Inc., **Año publicación:** 2016, **Última visita:** Mayo 2016

4. **https://tools.ietf.org/html/rfc2251**

   Autor: M. Wahl, **Año publicación:** 1997, **Última visita:** Mayo 2016

5. **http://www.openldap.org/doc/admin24/access-control.html**

   Autor: OpenLDAP Foundation, **Año publicación:** 2011, **Última visita:** Mayo 2016

6. **http://www.kouti.com/tables/userattributes.htm**

   Autor: Sakari Kouti, **Año publicación:** 2002, **Última visita:** Mayo 2016

7. Desarrollo e implementación de una oficina de información virtual

   Autor: Alejandro Abascal Crespo, **Año publicación:** 2015, **Última visita:** Mayo 2016

[REST]

8. https://faavoo.wordpress.com/2014/01/31/webservice-restful-en-java-usando-eclipse/

   Autor: Favo, **Año publicación:** 2014, **Última visita:** Junio 2016

9. http://asiermarques.com/2013/conceptos-sobre-apis-rest/

   Autor: Asier, **Año publicación:** 2013, **Última visita:** Junio 2016

10. http://www.restapitutorial.com/httpstatuscodes.html

    Autor: RestApiTutorial.com, **Año publicación:** - , **Última visita:** Junio 2016

[API]

11. https://docs.ldap.com/ldap-sdk/docs/javadoc/index.html

    Autor: UnboundID Corp, **Año publicación:** 2016, **Última visita:** Junio 2016

12. **http://jsonviewer.stack.hu/**

Autor: Gabor Turi**, Año publicación: - , Última visita:** Junio 2016

13. **https://carlosazaustre.es/blog/que-es-la-autenticacion-con-token/**

    Autor: Carlos Azaustre**, Año publicación:** 2015**, Última visita:** Junio 2016

14. **https://connect2id.com/products/nimbus-jose-jwt**

    Autor: Connect2id Ltd.**, Año publicación:** 2016**, Última visita:** Junio 2016

[WEB]

15. **https://secure.php.net/manual/es/index.php**

    Autor: PHP Group**, Año publicación:** 2016**, Última visita:** Junio 2016

16. **http://alvinalexander.com/php/php-curl-examples-curl_setopt-json-rest-web-service**

    Autor: Alvin Alexander**, Año publicación:** 2016**, Última visita:** Junio 2016

17. **https://www.w3schools.com/html/html_forms.asp**

    Autor: W3Schools**, Año publicación:** 1999-2017**, Última visita:** Junio 2016

[SEGURIDAD]

18. **http://www.htmlpoint.com/apache/10.htm**

    Autor: HTMLPOINT**, Año publicación:** 2006**, Última visita:** Julio 2016

19. **http://lacasadeljota.blogspot.com.es/2007/10/creacin-de-un-webservice-seguro.html**

    Autor: Juan Pablo**, Año publicación:** 2007**, Última visita:** Julio 2016

20. **http://www.akadia.com/services/ssh_test_certificate.html**

    Autor: Akadia**, Año publicación:** - **, Última visita:** Julio 2016

21. **https://www.123-reg.co.uk/support/answers/SSL-Certificates/Generate-a-CSR/generate-a-csr-apache-open-ssl-634/**

    Autor: 123 Reg**, Año publicación:** 2016**, Última visita:** Julio 2016

[APP]

22. **http://www.appinventor.org/**

    Autor: University of San Francisco**, Año publicación:** 2009-2017**, Última visita:** Septiembre 2016

23. **https://www.tutorialspoint.com/android/android_list_view.htm**

    Autor: tutorialspoint**, Año publicación:** - **, Última visita:** Septiembre 2016

24. **https://developer.android.com/training/articles/security-ssl.html#SelfSigned**

Autor: Google**, Año publicación:** 2008-2017**, Última visita:** Septiembre 2016

7 – Anexo 1: Documentación del API Rest

En este documento vamos a explicar las funciones disponibles en el API así como su uso.

Todas las peticiones y respuestas estarán en formato JSON. Los errores se mostraran como errores HTML.

1 – Login

- Resource URL: /LDAPserver/rest/

- Method: HTTP POST

Esta función será el punto de acceso al API. Tomará como argumentos el usuario y el password y nos devolverá el token correspondiente para usar en las operaciones que hagamos con el API. El token tendrá una duración máxima de una hora, tras lo cual habrá que solicitar un nuevo token.

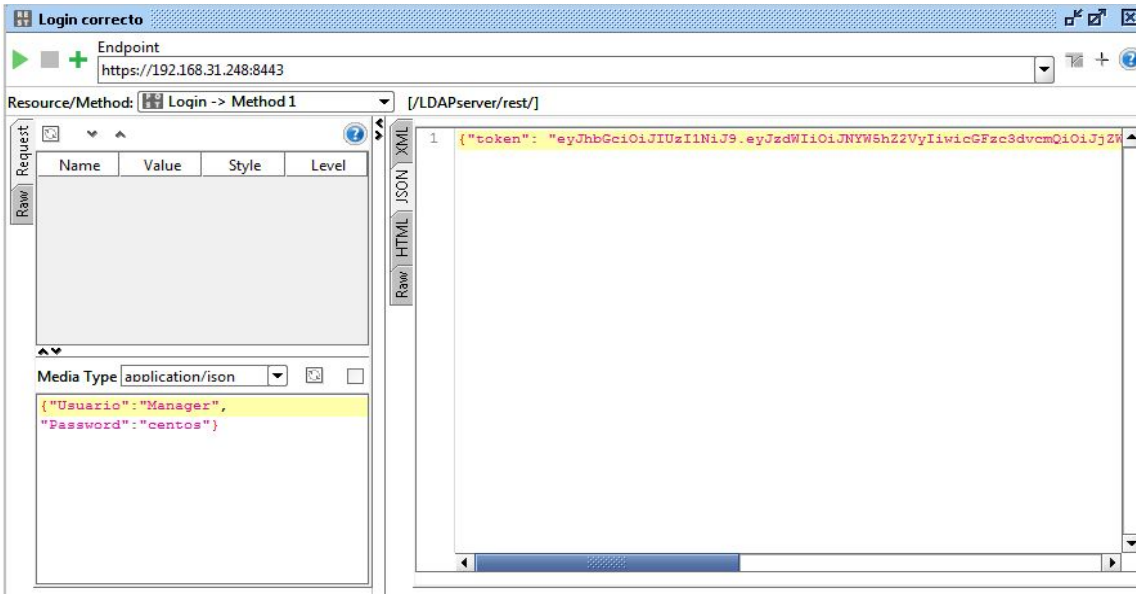

Figura 88: Login

2 – Búsqueda

- Resource URL: /LDAPserver/rest/search

- Method: HTTP POST

Esta función nos permite buscar una entrada en el directorio LDAP. Hay que indicarle el dato a buscar así como el tipo de entrada que estamos buscando (persona, edificio, asignatura, etc) y el token. En caso de no aportar el token, podremos seguir consultando datos del directorio pero solo aquellos que sean públicos.

Se mostrarán todos los resultados que contengan el dato a buscar independientemente de la posición (inicio, final, o en medio del nombre de la entrada).

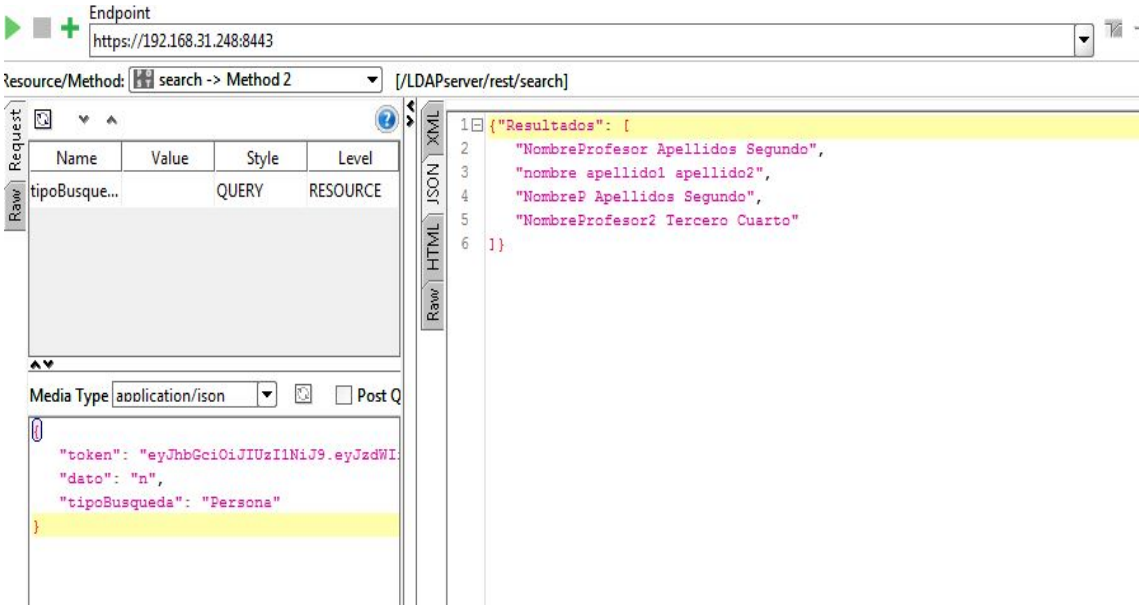

Figura 89: Búsqueda de personas

En el caso de la búsqueda de personas el dato a buscar acepta formato:

"dato" =" palabra1" → palabra1 representa el nombre de la persona a buscar

"dato" =", palabra2" → palabra2 representa el apellido de la persona a buscar, puede ser el primer o el segundo apellido de la persona

"dato" ="palabra1, palabra2" → se buscara una persona con nombre palabra1 y apellido palabra2

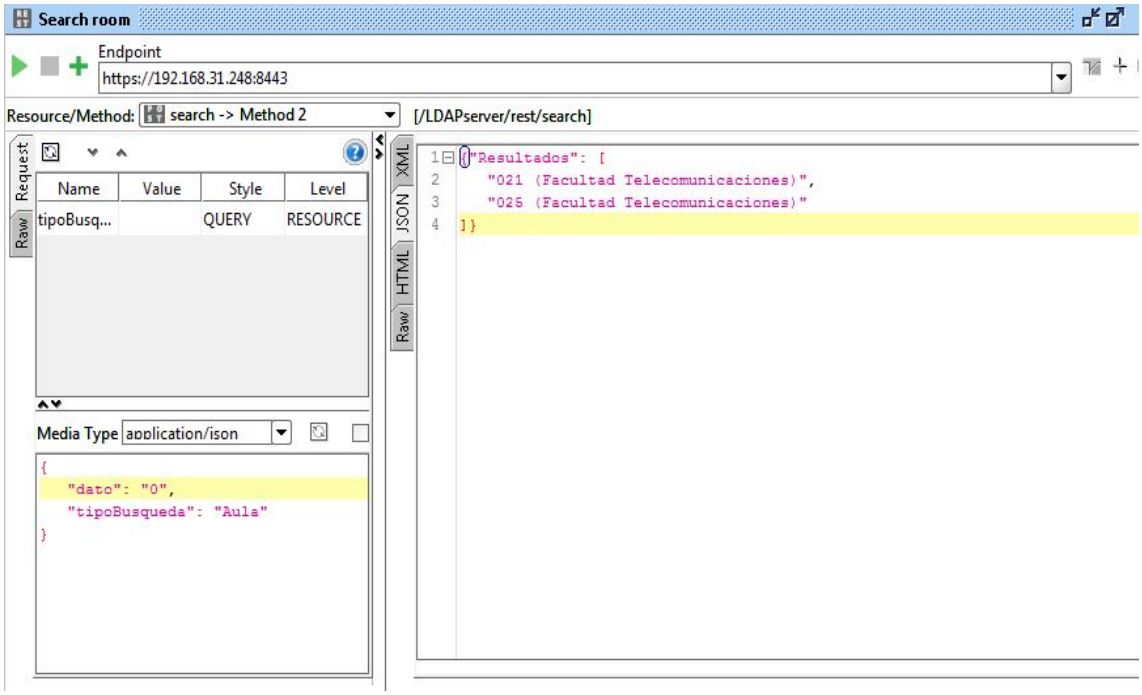

Figura 90: Búsqueda de aulas

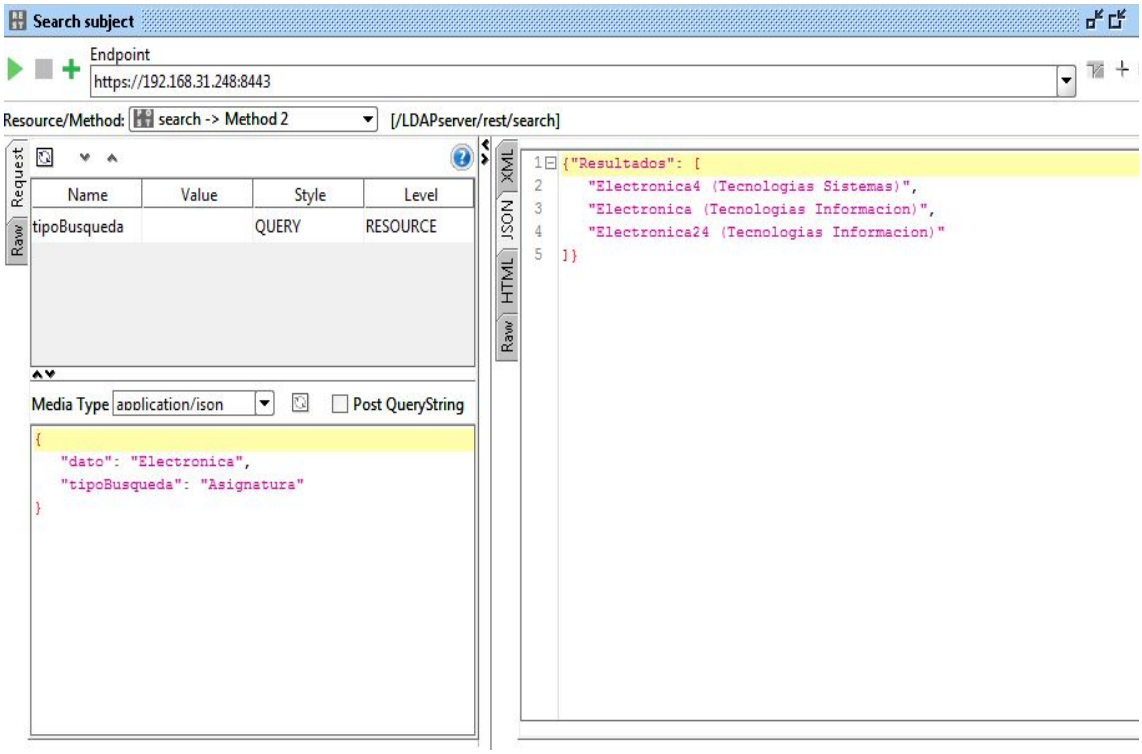

Figura 91: Búsqueda de asignaturas

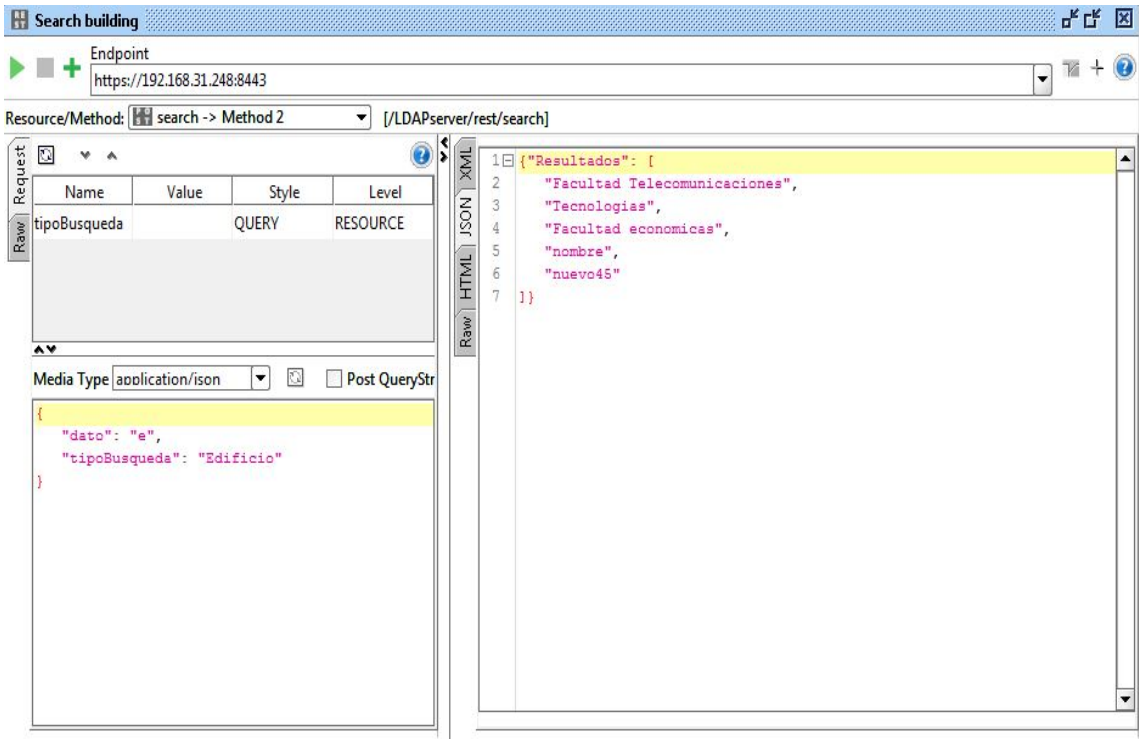

Figura 92: Búsqueda de edificios

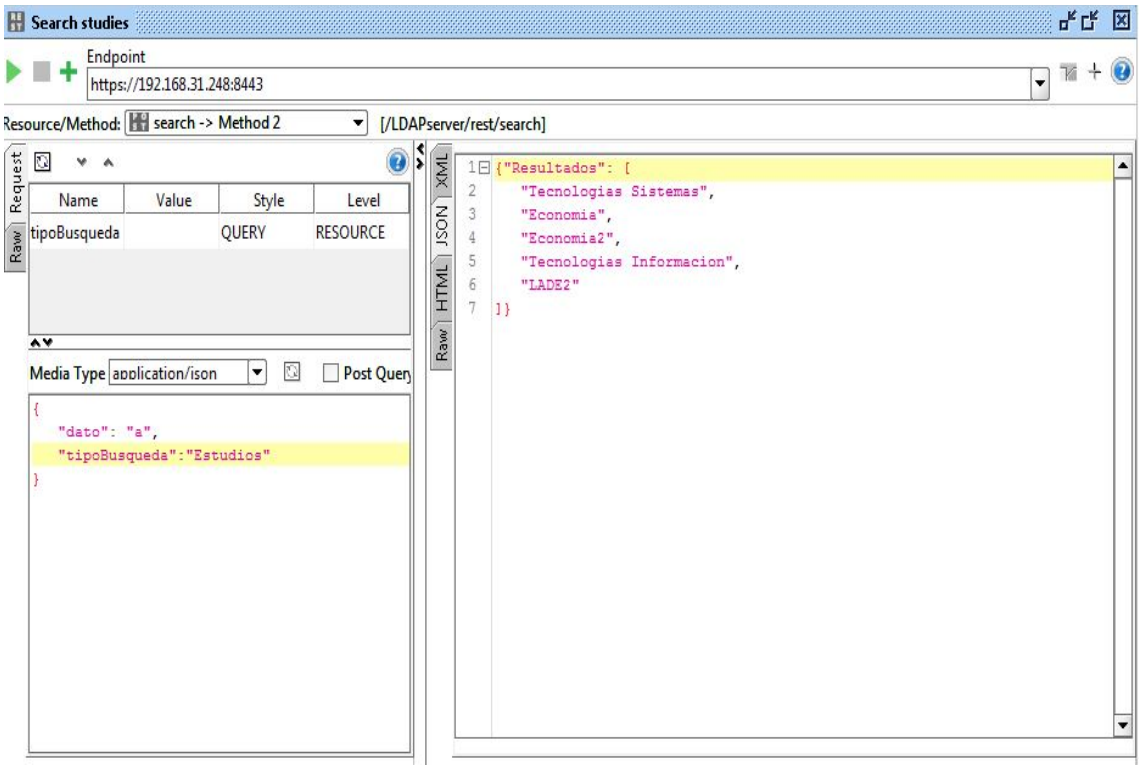

Figura 93: Búsqueda de estudios

Si en el dato a buscar no introducimos nada (string vacío) se mostrarán todos los resultados existentes en el sistema para ese tipo de entrada:

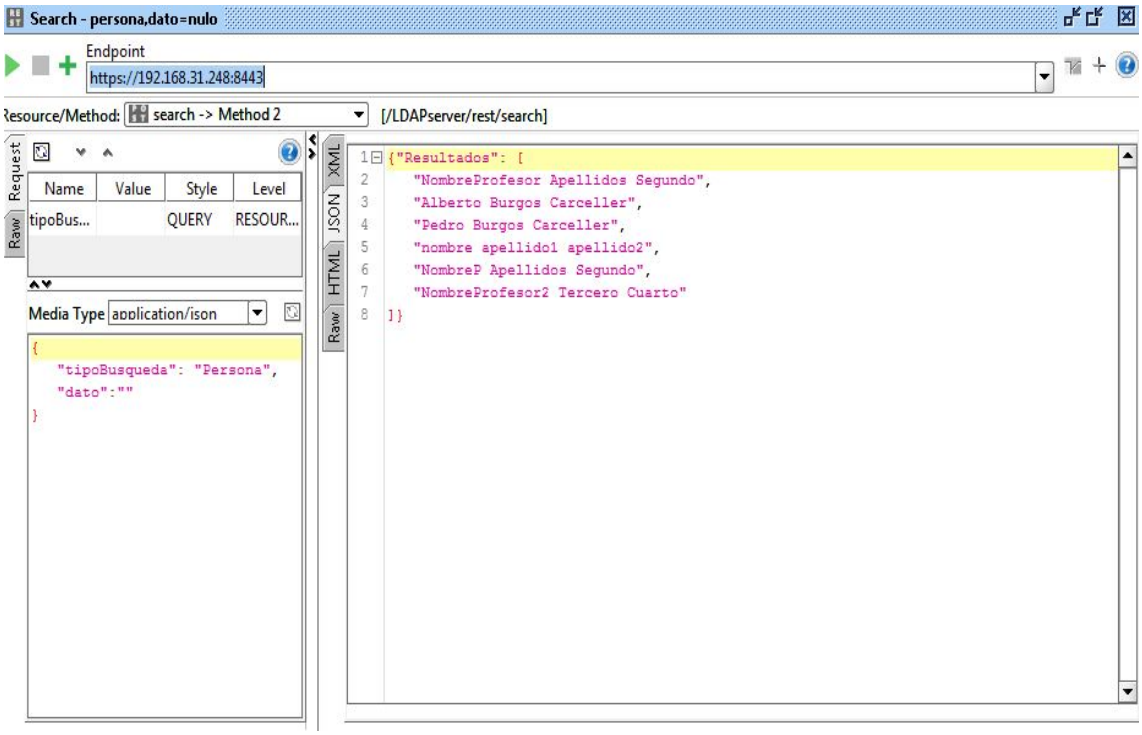

Figura 94: Búsqueda de personas. Todas las existentes en el sistema

3 – Usuario

- Resource URL: /LDAPserver/rest/user

- Method: HTTP POST

Esta función nos permite obtener todos los datos de un usuario en concreto que indiquemos en los campos “Nombre”,”Apellido1” y “Apellido2”.Si solo le pasamos el token de nuestra sesión, nos devolverá todos los datos de nuestra ficha de usuario. Si al solicitar los datos de un usuario, no le pasamos ningún token, solo nos devolverá los datos públicos del usuario solicitado.

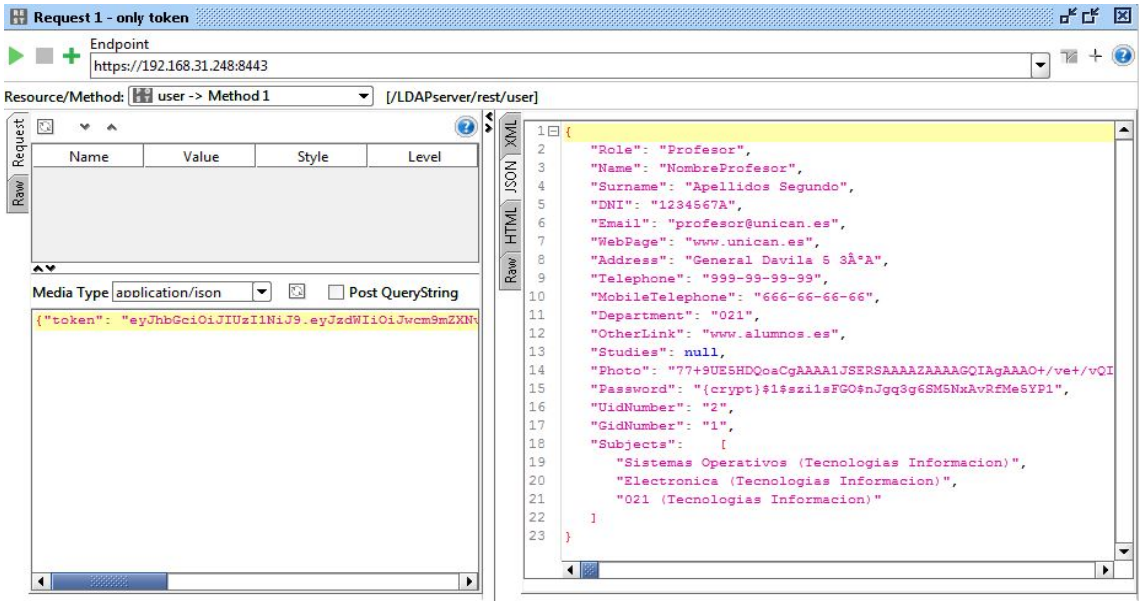

Figura 95: Solo token - devuelve nuestra ficha

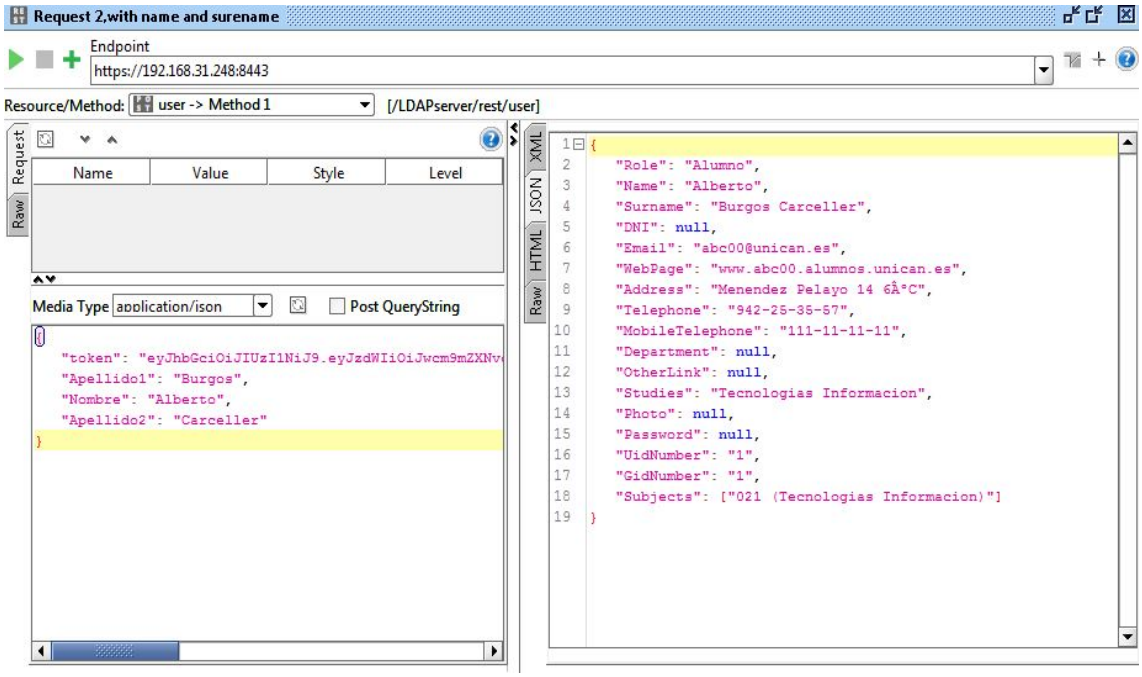

Figura 96: Datos de un usuario

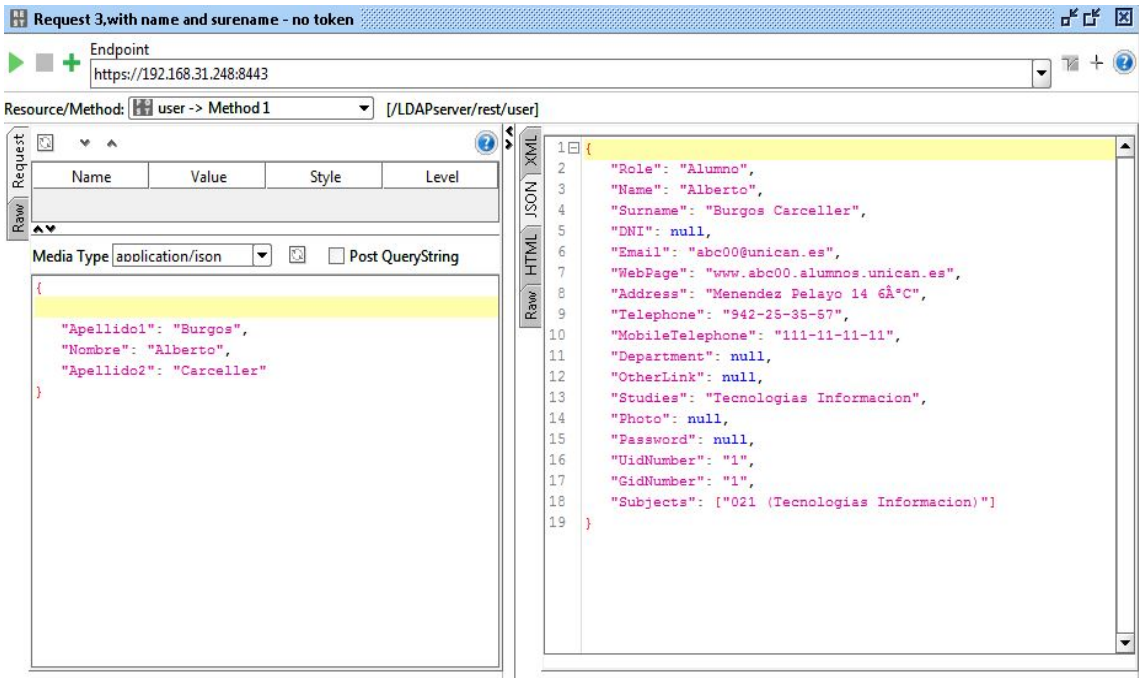

Figura 97: Datos de usuario, sin token - solo devuelve los datos públicos

4 – Aula

- Resource URL: /LDAPserver/rest/room

- Method: HTTP POST

En esta función podemos pedir todos los datos referentes a un aula indicando el aula que queremos en los campos “Aula” y “Edificio”. Si al solicitar los datos de un aula, no le pasamos ningún token, solo nos devolverá los datos públicos del aula solicitada.

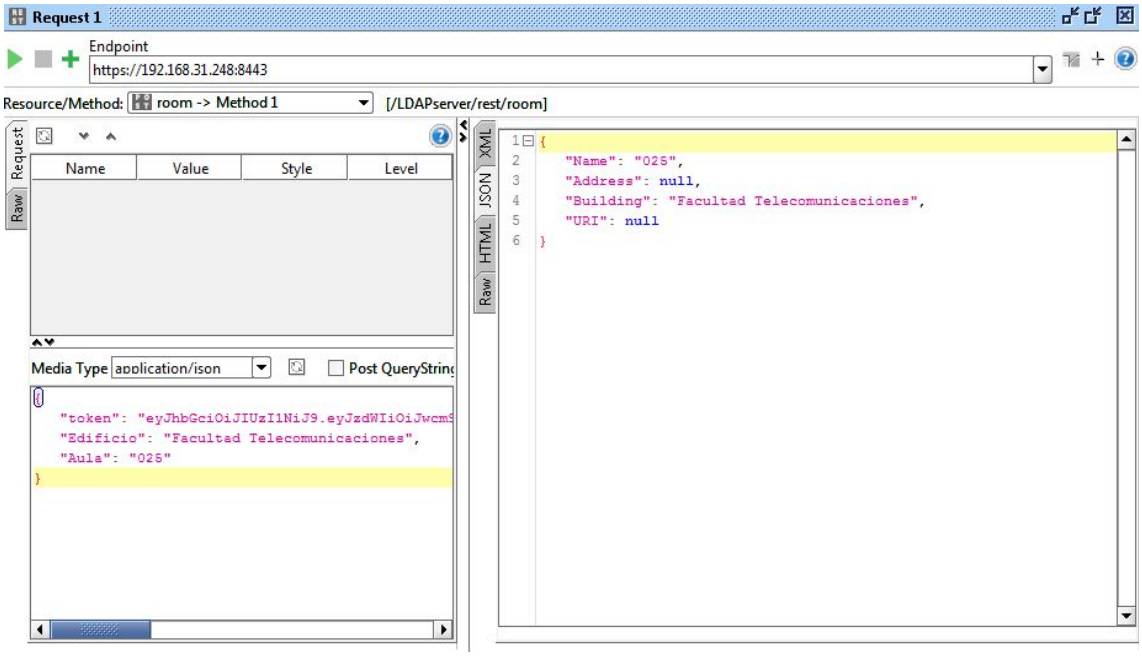

Figura 98: Datos aula

5 – Asignatura

- Resource URL: /LDAPserver/rest/subject

- Method: HTTP POST

En esta función podemos pedir todos los datos referentes a una asignatura indicando la asignatura que queremos en los campos "Asignatura" y "Estudios". Si al solicitar los datos de una asignatura, no le pasamos ningún token, solo nos devolverá los datos públicos de la asignatura solicitada.

Tenemos varios datos que se devuelven en función del usuario que los pida:

- ➔ Si el usuario es uno de los profesores responsables de la asignatura tendrá acceso a todos los datos de la asignatura, incluida la lista de los alumnos con su DNI correspondiente.
- ➔ Si el usuario es uno de los alumnos de la asignatura tendrá acceso a todos los datos de la asignatura excepto a la lista de alumnos (debido a la ley de protección de datos).
- ➔ Si el usuario es alguien ajeno a la asignatura (ni profesor ni alumno) no tendrá acceso ni a la lista de alumnos ni a la URL de la página web de la asignatura (el campo "WebPage").

Hay dos enlaces que contendrán la información de la asignatura:

- Guide → contendrá la información de la guía académica de la asignatura. Como la guía es pública este enlace será visible por todos los usuarios.

- WebPage → este será un enlace a la página web de la asignatura (página web propia, moodle, aula virtual, etc) donde el alumno tendrá toda la información de sus notas y trabajos. Como esta información es privada, solo los alumnos y profesores podrán ver este enlace.

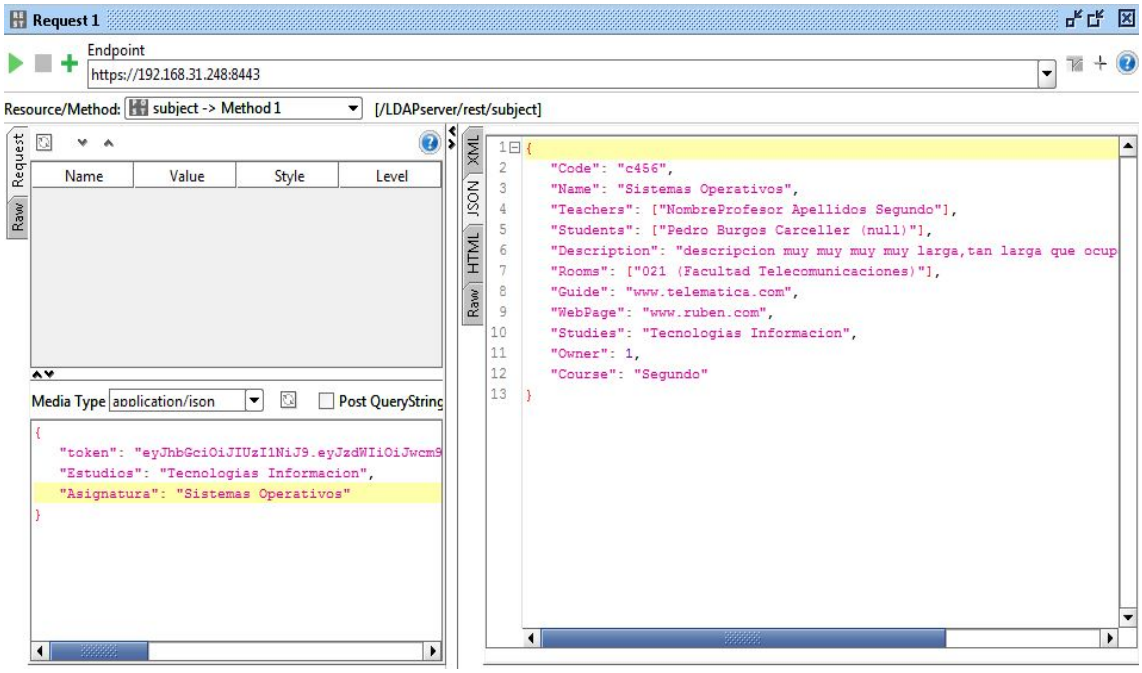

Figura 99: Datos de la asignatura - usuario = profesor

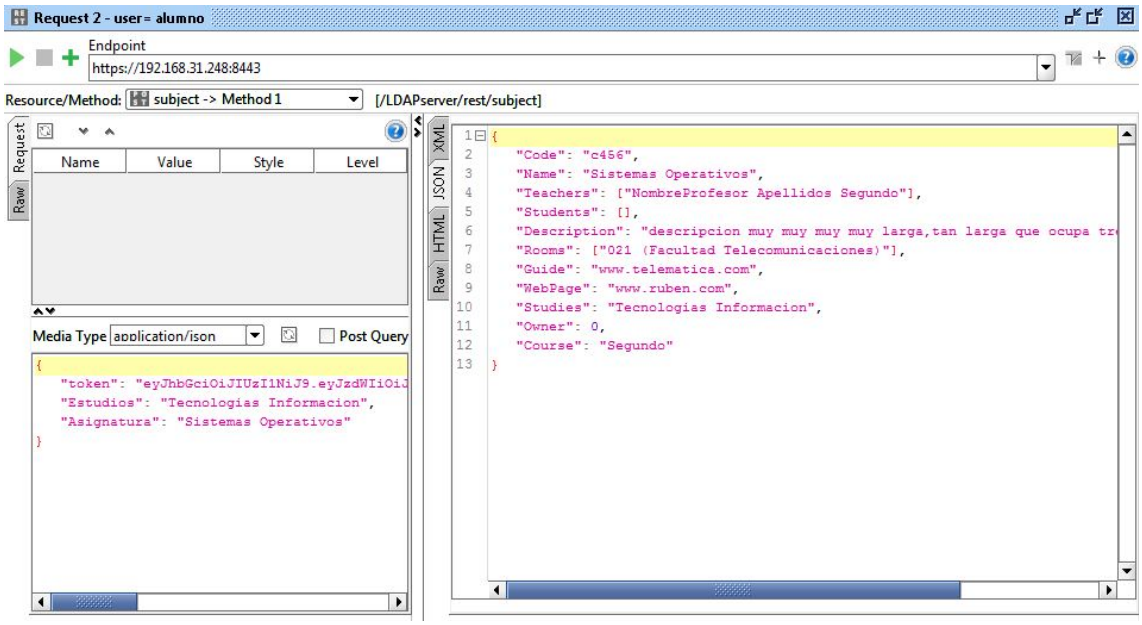

Figura 100: Datos de la asignatura - usuario = alumno

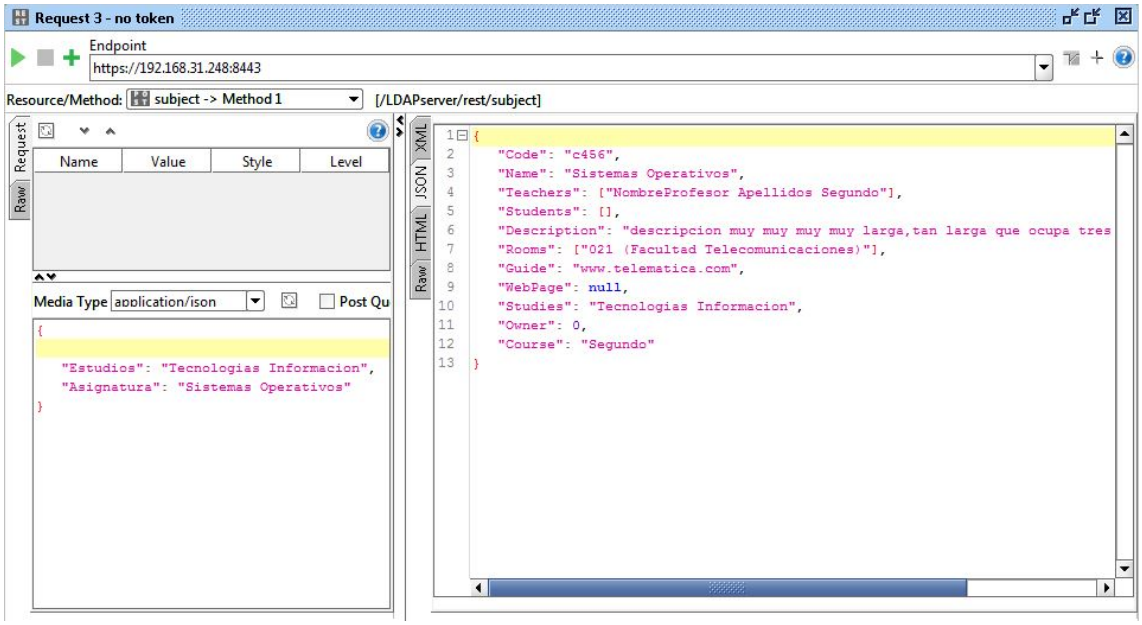

Figura 101: Datos de la asignatura - datos públicos

6 – Edificio

- Resource URL: /LDAPserver/rest/building

- Method: HTTP POST

En esta función podemos pedir todos los datos referentes a un edificio indicando el edificio que queremos en el campo “Edificio”. Si al solicitar los datos de un edificio, no le pasamos ningún token, solo nos devolverá los datos públicos del edificio solicitado.

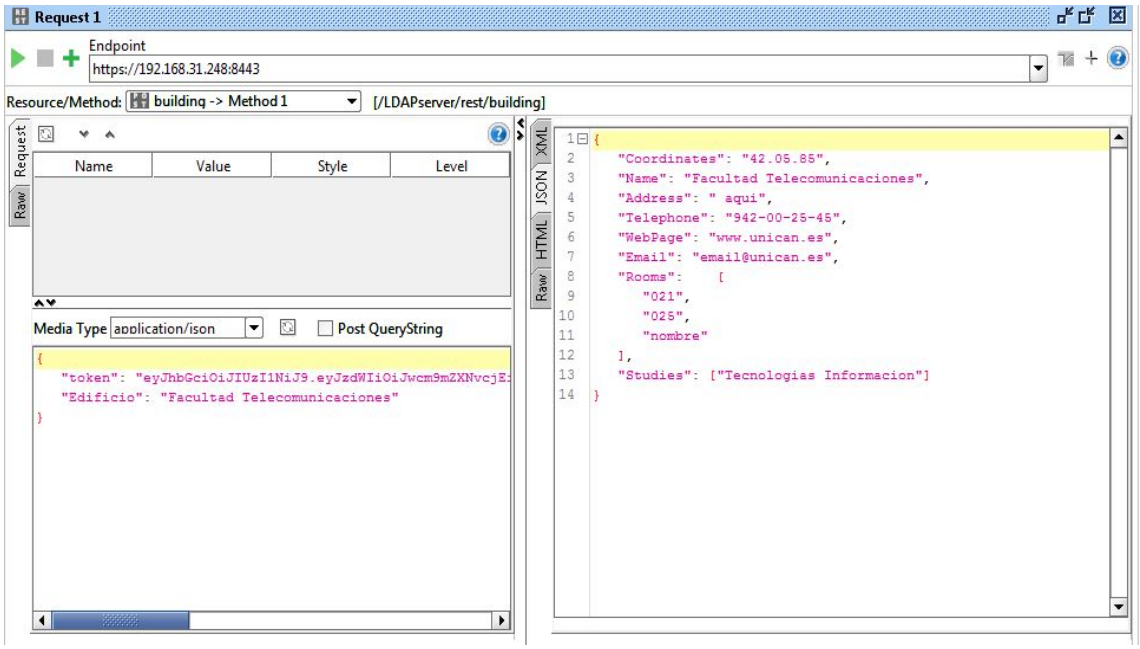

Figura 102: Datos de un edificio

7 – Estudios

- Resource URL: /LDAPserver/rest/studies

- Method: HTTP POST

En esta función podemos pedir todos los datos referentes a unos estudios indicando los estudios que queremos en el campo “Estudios”. Si al solicitar los datos de unos estudios, no le pasamos ningún token, solo nos devolverá los datos públicos de los estudios solicitados.

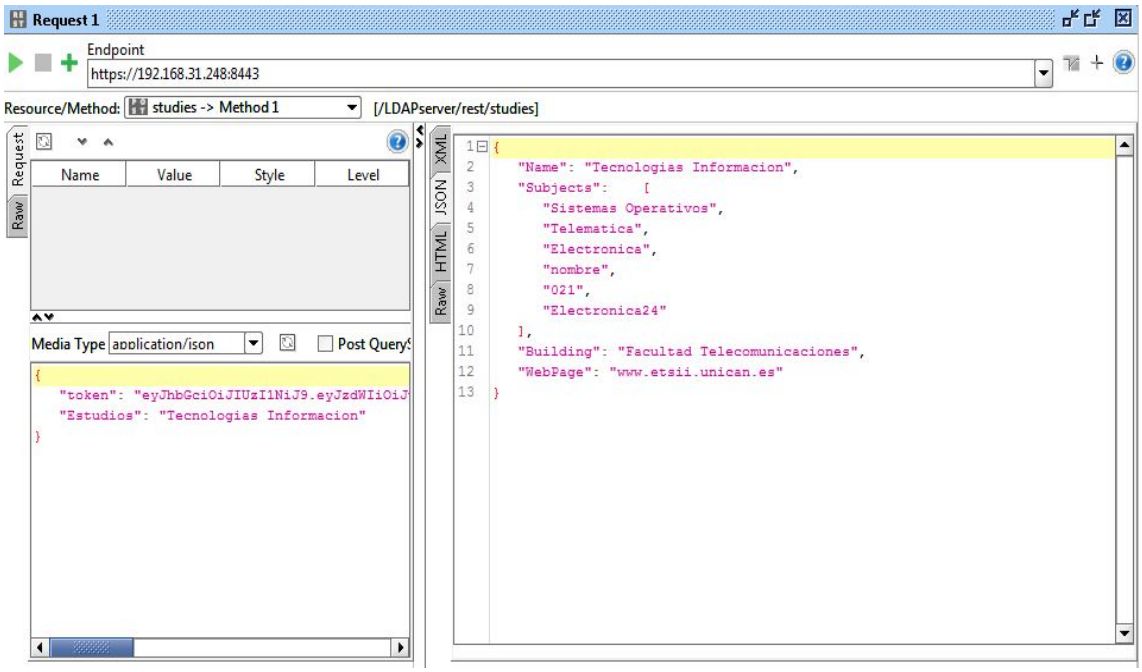

Figura 103: Datos de unos estudios

8 – Crear entradas

- Resource URL: /LDAPserver/rest/entry/new

- Method: HTTP POST

Esta función es una de las más complejas e indispensable. Nos va a permitir crear cualquier tipo de entrada en el directorio. En función del tipo de entrada que queramos crear vamos a necesitar introducir más o menos información, parte de la cual será obligatoria.

Como esta es una operación de gestión, solo el manager del directorio (usuario "Manager") tendrá los permisos necesarios para crear nuevas entradas.

Independientemente de la entrada que vayamos a crear debemos pasarle dos argumentos:

- Token → token del usuario que realiza la acción

- TipoEntrada → tipo de entrada que queramos crear. Podrá tomar los siguientes valores:

"Persona","Asignatura","Aula","Estudios","Edificio"

8.1 – Crear usuarios

Campo	***Requerido***	***Descripción/Observaciones***
Usuario	Si	Se usará el mismo usuario que en el campus virtual (se puede cambiar)
Password	Si	Se usará el mismo usuario que en el campus virtual (se puede cambiar)
Nombre	Si	
Apellido1	Si	
Apellido2	Si	
Role	Si	Puede tomar los valores "Alumno" o "Profesor"
Estudios	?	Solo es requerido si el Role = Alumno
DNI	Si	
Email	Si	
gidNumber	No	N. º de grupo. Solo usado si el directorio funciona como método de inicio de sesión en equipos.
uidNumber	No	N. º de usuario. Solo usado si el directorio funciona como método de inicio de sesión en equipos.
Web	No	Enlace a web personal
Telephone	No	Teléfono casa/oficina/departamento

Mobile	No	Teléfono móvil persona
Photo	No	Foto en base64
Address	No	Dirección casa/departamento
Department		N.º departamento – también se puede usar como enlace a la ficha de un aula/departamento
OtherLink	No	Enlace al curriculum/expediente del alumno/profesor

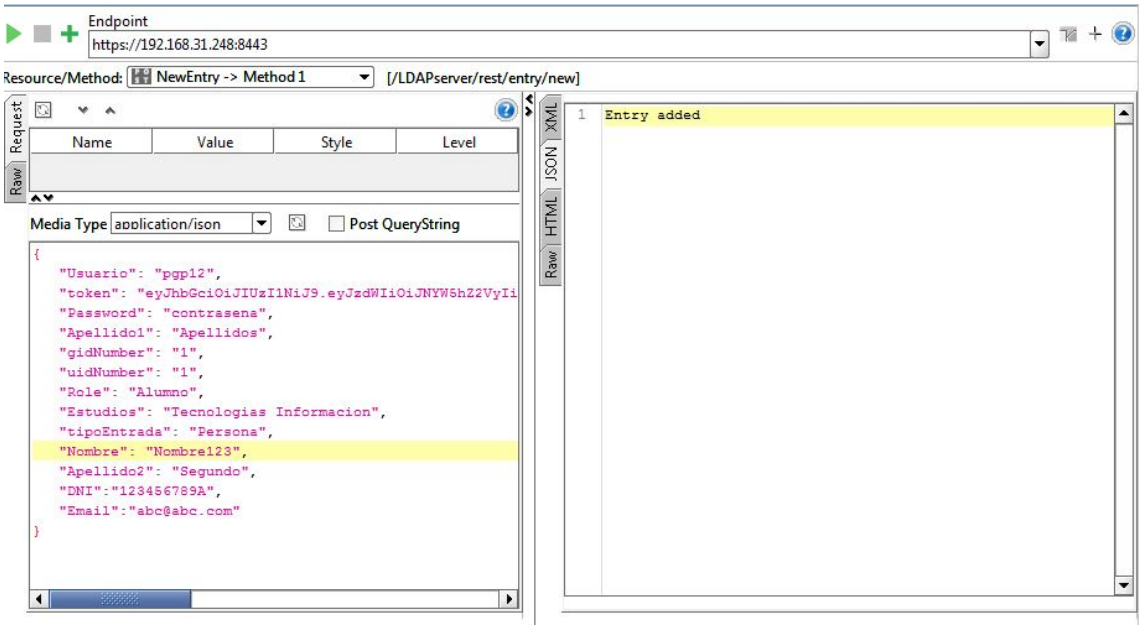

Figura 104: Nuevo usuario - información mínima necesaria

8.2 – Crear asignatura

Campo	***Requerido***	***Descripción/Observaciones***
Codigo	No	Código de la asignatura
Curso	No	Curso al que corresponde la asignatura
Teachers	No	Profesores responsables de la asignatura
Students	No	Alumnos de la asignatura
Descripcion	No	Descripción de la asignatura
Web	No	Enlace al documento con la guía académica
OtherLink	No	Enlace a web donde este almacenado el material de la

		asignatura
Rooms	No	Aulas en que se imparte la asignatura
Estudios	Si	Plan de estudios al que pertenece la asignatura
Asignatura	Si	Nombre de la asignatura

Al crear una asignatura, para que se hagan efectivos sus permisos se producirá un reinicio del servidor LDAP. Se realizará de forma automática al introducir la asignatura, no requerirá acción por parte del usuario.

Al crear la asignatura no se podrán introducir ni los profesores responsables de la asignatura, ni los alumnos ni las aulas. Esto se deberá hacer a posteriori usando la función de modificar o de importar alumnos debido a que se creará un enlace a la entrada de la persona/aula en cuestión y para ello debemos asegurarnos de que estas entradas existen.

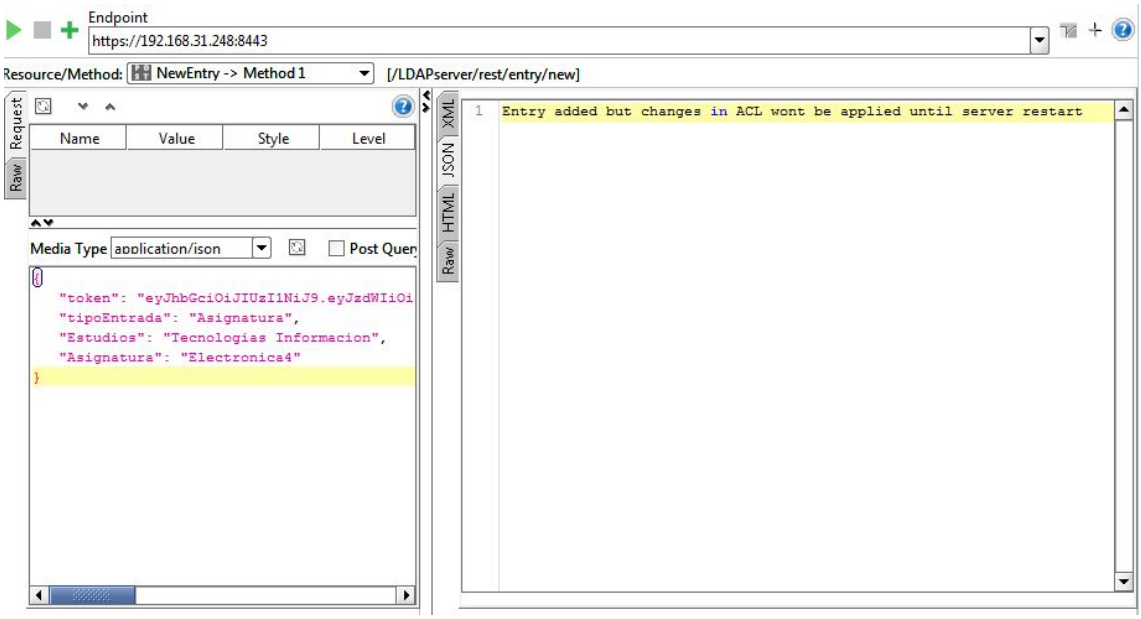

Figura 105: Nueva asignatura - Información mínima necesaria

8.3 – Crear aula

Campo	***Requerido***	***Descripción/Observaciones***
Address	No	Dirección del aula
Coordenadas	No	URL que indique la localización.
Edificio	Si	Edificio en el que se sitúa el aula
Aula	Si	Nombre del aula

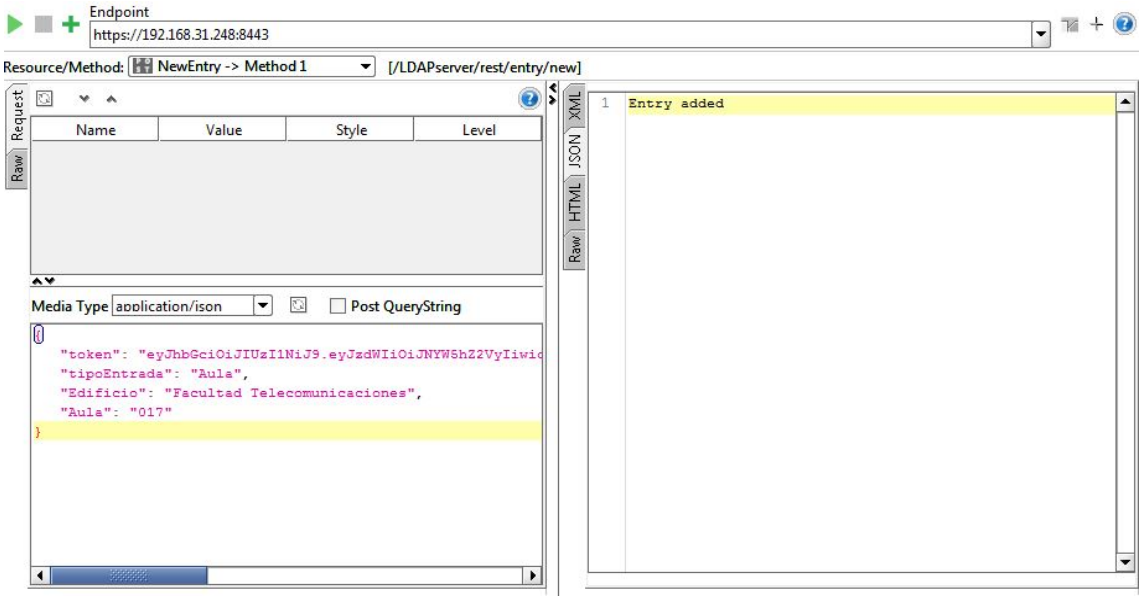

Figura 106: Nuevo Aula - Información mínima necesaria

8.4 – Crear estudios

Campo	***Requerido***	***Descripción/Observaciones***
Address	No	Dirección del edificio
Web	No	URL de la web del plan de estudios
Edificio	No	Enlace a la ficha del edificio en el que se imparten los estudios
Estudios	Si	Nombre de los estudios

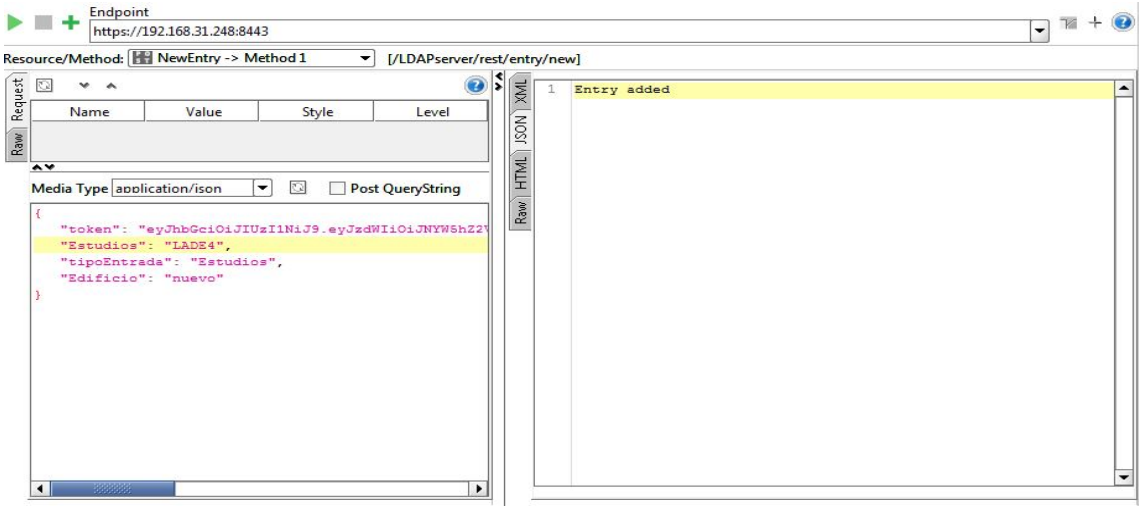

Figura 107: Nuevos Estudios - Información mínima necesaria

8.5 – Crear edificio

Campo	***Requerido***	***Descripción/Observaciones***
Address	No	Dirección del edificio
Coordenadas	No	URL que indique la localización.
Edificio	Si	Nombre del edificio
Telephone	No	Teléfono asociado al edificio
Email	No	Email asociado al edificio
Web	No	URL asociada al edificio

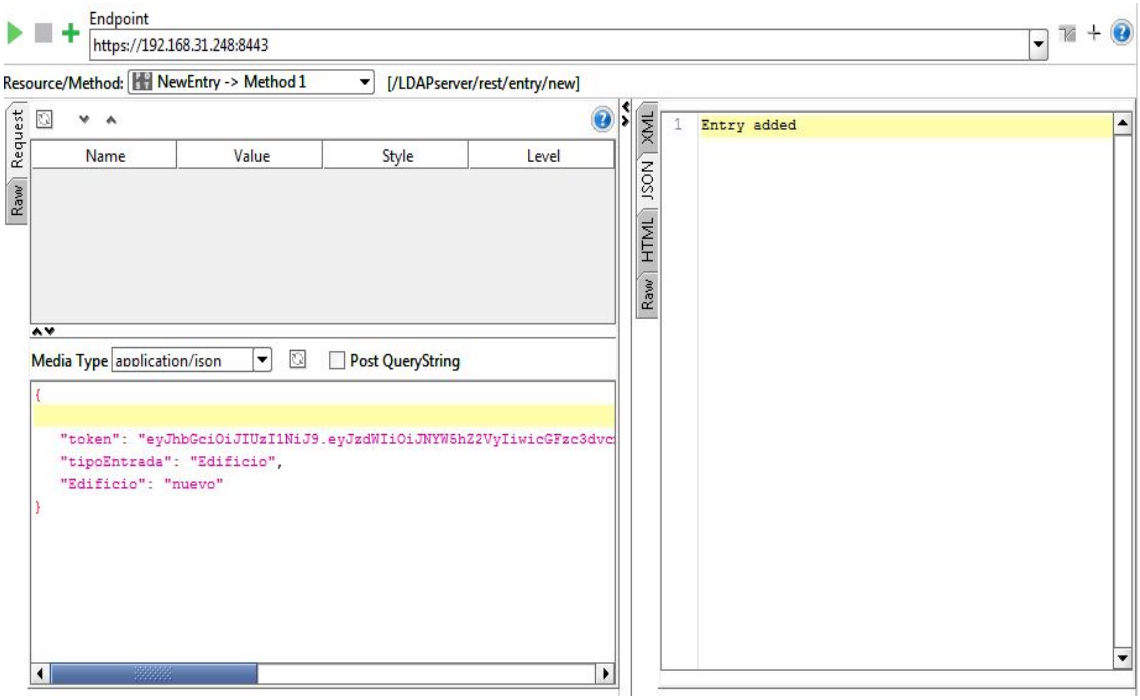

Figura 108: Nuevo Edificio - Información mínima necesaria

9 – Eliminar entradas

- Resource URL: /LDAPserver/rest/entry/delete

- Method: HTTP POST

Esta función nos permite borrar una entrada. Como medida de seguridad, solo el usuario Manager tendrá permitido borrar entradas.

Para borrar una entrada tenemos que indicar el tipo de entrada que queremos borrar así como los datos identificativos de la entrada.

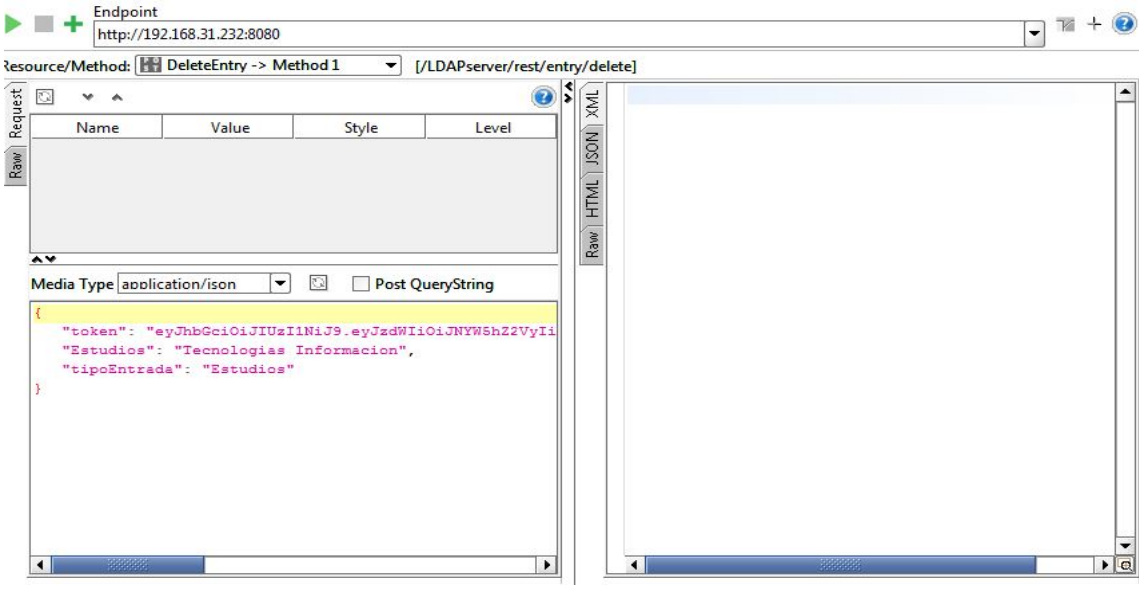

Figura 109: Borrar entrada perteneciente a un plan de estudios

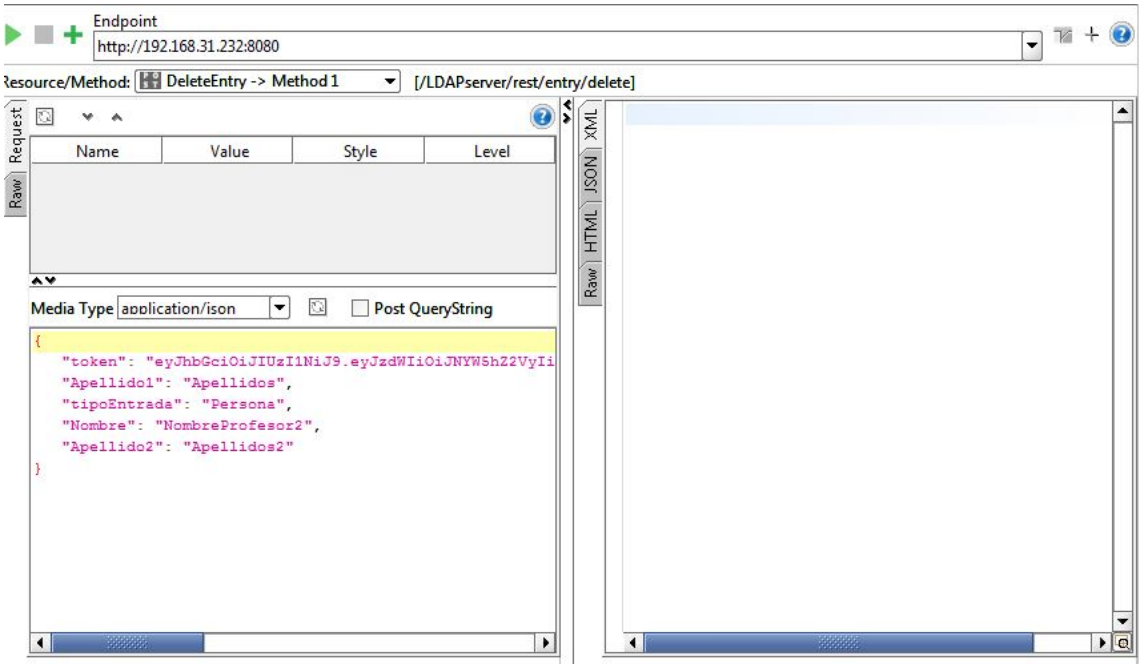

Figura 110: Borrar entrada perteneciente a un usuario

10 – Modificar entradas

- Resource URL: /LDAPserver/rest/entry/modify

- Method: HTTP POST

Esta función nos permitirá cambiar un dato de una entrada por otro dato nuevo. Funciona en base a los permisos creados en el directorio, es decir, el usuario Manager podrá modificar cualquier dato, y el resto de usuarios solo podrán modificar algún dato de su entrada, no la de otros usuarios.

Algunos datos se ha evitado su modificación (como los datos identificativos de una persona) para evitar problemas luego, cuando los datos no coincidan. En caso de necesitar cambiarse alguno de esos datos, deberá hacerlo el Manager.

Debemos introducir los siguientes argumentos para modificar algún dato:

- "tipoEntrada" → tipo de entrada sobre la que vamos a actuar (persona, aula, asignatura,..)
- "tipoAtributo" → nombre del atributo que queremos modificar (nombre, código,…)
- "valor" → nuevo valor que queramos que tome el atributo
- "tipoModificación" → tipo de modificación que queramos realizar

  → podrá tomar tres valores:

  - "REPLACE"
  - "ADD"
  - "DELETE"

Aparte de estos datos debemos introducir los datos identificativos de la entrada que queramos modificar (nombre y apellidos en el caso de personas, nombre y edificio en el caso de aula, etc).

El tipo de modificación a usar depende de lo que queramos hacer y si ese dato antes tenía valor o no:

- "REPLACE" → el dato a modificar antes tenía un valor y queremos poner otro valor
- "ADD" → el dato a modificar antes no tenía valor y queremos ponerle uno
- "DELETE" → el dato a modificar tenía un valor y ahora queremos que no tenga ninguno

Esta restricción es debida a la biblioteca que usamos para manejar el directorio LDAP, ya que si usábamos un valor incorrecto en el tipo de operación con un valor vacío, podía provocar un comportamiento inesperado. De esta forma, aunque es más incómodo, nos asegurábamos de la operación que realizamos.

El comportamiento inesperado es que podía provocar que un dato que solo debe tener un valor (aunque el directorio permita que tenga varios) tuviera más de un valor, lo cual podría provocar fallos a la hora de modificar o acceder a esos datos. (Ver ModificationType)

Existe un caso especial a mencionar que es que si ponemos como valor "All" en una asignatura y con tipoModificación=DELETE, borraremos todos los alumnos de una asignatura en una sola operación.

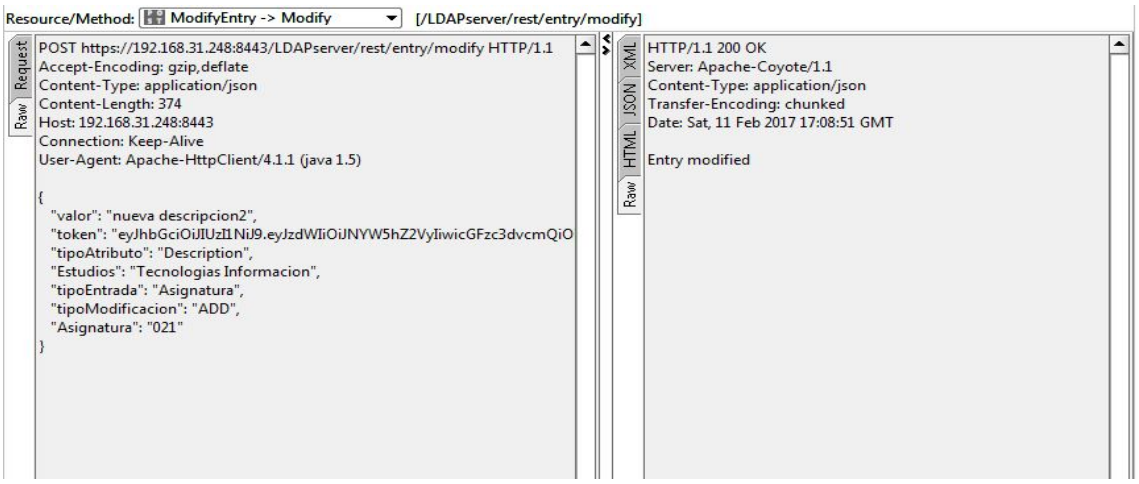

Figura 111: Modificación de la descripción de una asignatura

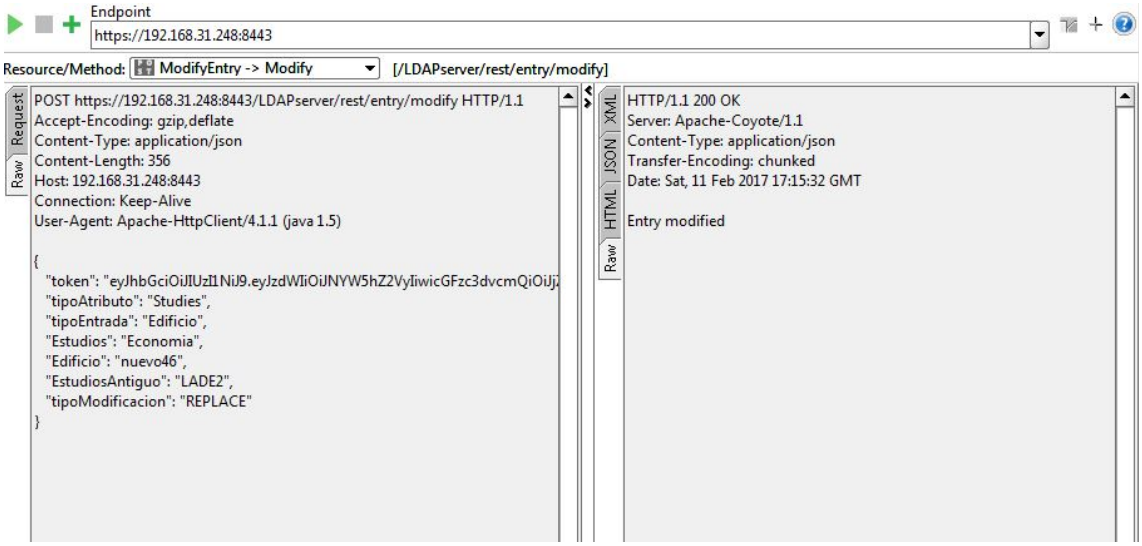

Figura 112: Modificación de los estudios de un edificio

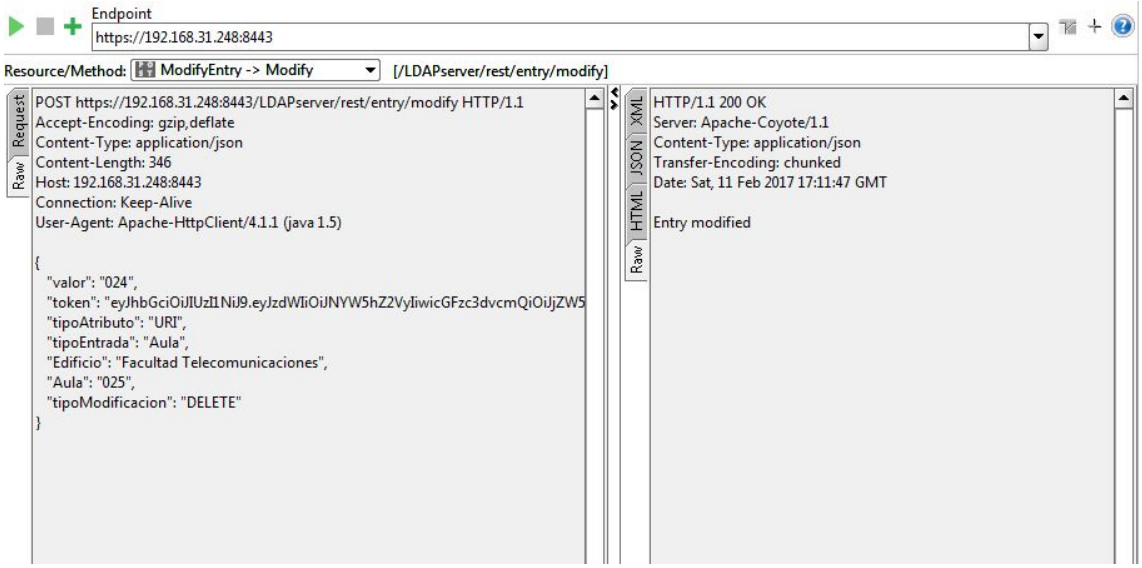

Figura 113: Modificación del URI de un aula

11 – Exportar entradas (backup)

- Resource URL: /LDAPserver/rest/entry/export

- Method: HTTP POST

Esta función nos va a permitir exportar la información del directorio a un archivo de texto plano, a partir del nodo que le indiquemos. Se recorrerá el árbol desde ese nodo hacia abajo. Esta función hace las veces de backup del directorio. Solo el Manager puede realizar esta función.

Tendremos que pasarle dos argumentos:

- "rutaArchivo" → ruta donde queremos guardar el archivo de backup
- "TipoNodoRaiz" → tipo de entrada a partir de la cual queremos hacer el backup (persona, aula, etc)
  → si ponemos el valor "Raiz" nos hará un backup de todo el directorio

Si queremos hacer el backup desde una entrada concreta, aparte del tipo de entrada, deberemos pasar los datos identificativos de esa entrada (nombre y apellidos, etc).

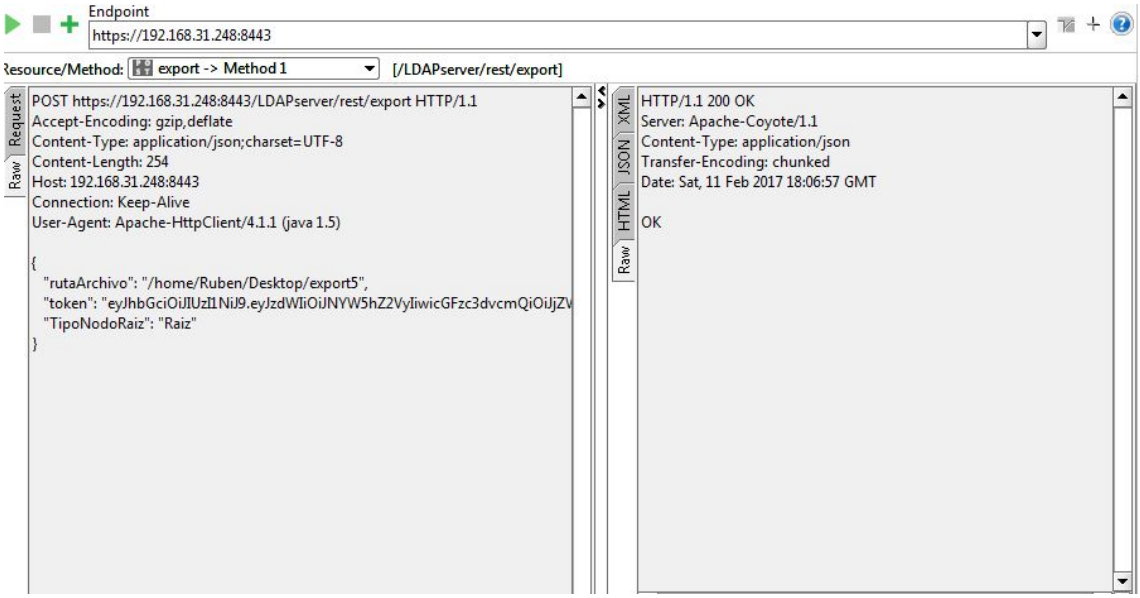

Figura 114: Backup de datos del directorio completo

```
dn: ou=estudios,dc=mydomain,dc=com
objectClass: top
objectClass: organizationalUnit
ou: estudios

dn: ou=edificios,dc=mydomain,dc=com
objectClass: top
objectClass: organizationalUnit
ou: edificios

dn: ou=profesores,dc=mydomain,dc=com
objectClass: top
objectClass: organizationalUnit
ou: profesores
```

Figura 115: Archivo de exportación

Como vemos el backup del directorio se hace en formato LDAP. Esto se hace así debido al tiempo que conllevaría traducirlo a lenguaje natural si el directorio es de gran tamaño, tanto al hacer el backup como al restaurarlo.

12 – Importar entradas (backup)

- Resource URL: /LDAPserver/rest/entry/import

- Method: HTTP POST

Esta función nos permite usar el backup del directorio para restablecer los datos. Solo puede ejecutarla el manager.

El único argumento que debemos pasarle es la ruta del archivo de backup.

El API no sobrescribe las entradas, por lo que habrá que borrar todo antes de importar.

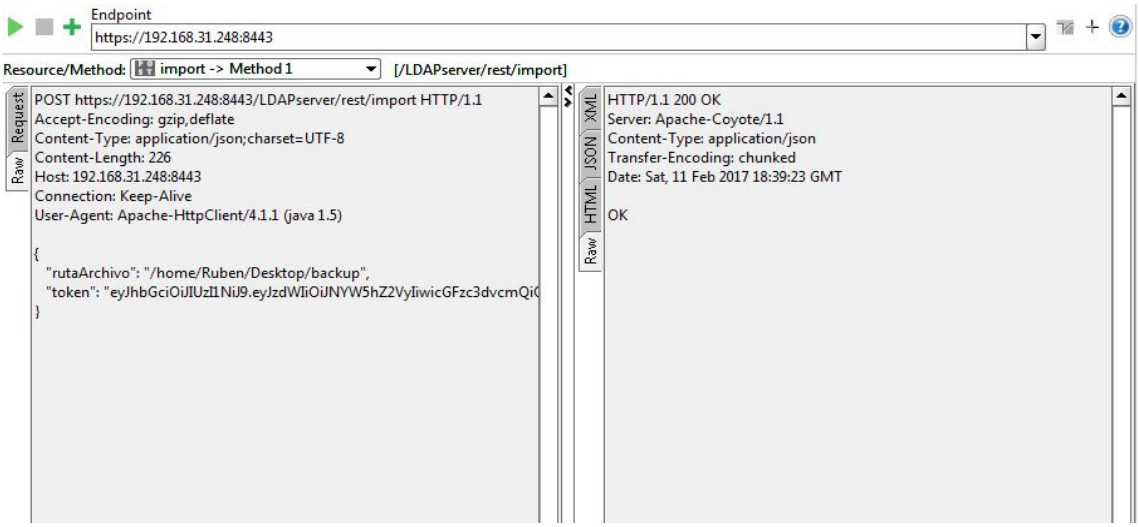

Figura 116: Restauración de backup

13 – Importar alumnos

- Resource URL: /LDAPserver/rest/entry/import/members

- Method: HTTP POST

Dado que la lista de alumnos de una clase es larga, no es práctico introducirlos uno a uno. Por ello se ha creado esta función especial que los introducirá todos de golpe a partir de un archivo de texto.

Le tenemos que pasar como argumento la ruta donde está el archivo con la lista así como los datos identificativos de la asignatura. Solo el manager puede realizar esta operación.

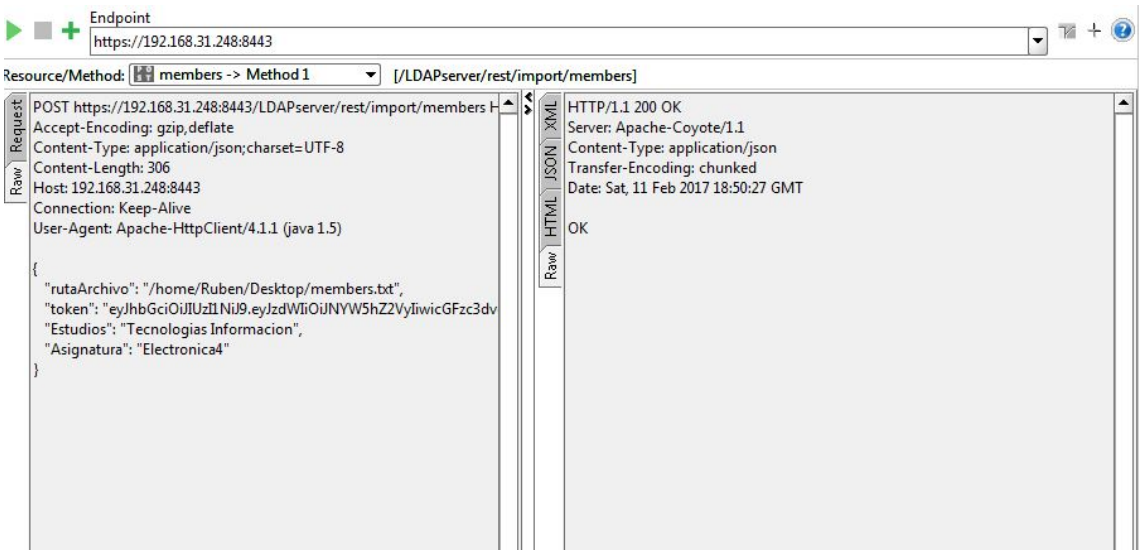

Figura 117: Adjuntar alumnos a una asignatura

El archivo conteniendo los nombres será un archivo de texto plano con el formato Nombre, Apellido1 Apellido 2

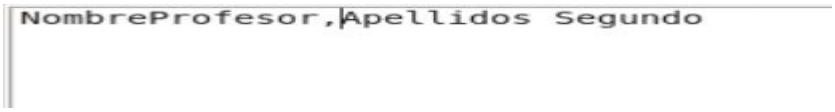

Figura 118: Archivo de alumnos

Si se produce un error, se indicarán cuantos alumnos de esa lista no han podido importarse.

7 – Anexo 2: Tablas equivalencia API-LDAP

En este apartado vamos a mostrar la equivalencia entre nuestras variables y su equivalente el LDAP así como el tipo de dato que almacenan.

- PERSONA

Campo API	***Campo LDAP***	***Tipo de dato***
Usuario	uid	String
Password	userPassword	Octect String
Nombre	givenName	String
Apellido1	sn	String
Apellido2	sn	String
Role	-	String(Se obtiene de la estructura del directorio)
Estudios	-	String(Se obtiene de la estructura del directorio)
DNI	employeeNumber	String
Email	mail	String
gidNumber	gidNumber	Integer
uidNumber	uidNumber	Integer
Web	labeledURI	String
Telephone	homePhone	String
Mobile	mobile	String
Photo	jpegPhoto	Octect String
Address	PostalAddress	String
Department	seeAlso	DistinguishedName (DN de una entrada)
OtherLink	o	String

- ASIGNATURA

Campo API	***Campo LDAP***	***Tipo de dato***
Codigo	businessCategory	String
Curso	ou	String
Teachers	owner	DistinguishedName (DN de una entrada)
Students	member	DistinguishedName (DN de una entrada)
Descripcion	description	String
Web	labeledURI	String
OtherLink	o	String
Rooms	seeAlso	DistinguishedName (DN de una entrada)
Estudios	-	String(Se obtiene de la estructura del directorio)
Asignatura	cn	String

- AULA

Campo API	***Campo LDAP***	***Tipo de dato***
Address	postalAddress	String
Coordenadas	labeledURI	String
Edificio	-	String (Se obtiene de la estructura del directorio)
Aula	cn	String

- ESTUDIOS

Campo API	***Campo API***	***Campo API***
Address	postalAddress	String
Web	labeledURI	String
Edificio	seeAlso	DistinguishedName (DN de una entrada)
Estudios	ou	String

- EDIFICIO

Campo API	***Campo API***	***Campo API***
Address	postalAddress	String
Coordenadas	destinationIndicator	String
Edificio	ou	String
Telephone	telephoneNumber	String
Email	businessCategory	String
Web	labeledURI	String