Universidad de La Habana
Facultad de Matemática y Computación

Sistema de extracción y publicación de información relacionada con recursos humanos de la Universidad de La Habana

Autor: Eric Nordelo Galiano

Tutor: **Lic. Gilberto García**

Tutor: **Lic. Darío Álvarez Arteaga**

Trabajo de Diploma
presentado en opción al título de
Licenciado en Ciencias de la Computación

Agosto de 2019

Agradecimientos

Para empezar, agradezco mucho a mi familia, sobre todo a mis padres, por su guía y su constante apoyo en el transcurso de mi licenciatura. A mi madre por priorizar mis estudios siempre, siendo el timón que nunca permitió que perdiera el rumbo aún en los tiempos más difíciles. A mi padre por enseñarme que la inteligencia que no se aprovecha no existe, por ser mi primer maestro y mi tutor en la tesis de la vida, por compartirme sus experiencias y su amor por el estudio.

Agradezco a todos mis profesores y maestros, desde las primeras enseñanzas, por motivarme siempre a aprender, por enseñarme el significado de la palabra dedicación, por todas las horas de su vida que me destinaron a mí y a tantos otros niños y jóvenes. A todos los profesores que aportaron su grano de arena al profesional que ya casi soy en estos cinco años de universidad, de entre los cuales quiero mencionar a tres con especial cariño: A Idania Urrutia, profesora de Análisis Matemático, por su disposición y afecto, por enseñarme a amar las matemáticas; a Alfredo Somoza, profesor de Matemáticas Discretas, por enseñarme a pensar; y por último pero no menos importante a Pedro Quintero Rojas, pqr como cariñosamente le llamamos, por ser como un segundo padre, profesor de Redes y Programación de Máquinas.

A mi tutor Gilberto García, por encontrar siempre el tiempo para mí, por ayudarme en todo el desarrollo de esta tesis con su conocimiento.

A mis compañeros de estudio, por hacer de estos cinco años los mejores de mi vida hasta hoy. A mis amigos con los que compartí el estudio, haciendo proyectos, practicando deportes, recibiendo clases o yendo a fiestas, gracias por hacer que estos años fueran realmente divertidos.

A mis amigos por apoyarme siempre, por su respaldo constante, por recordarme el valor de la amistad en los buenos y los malos momentos. En especial a Ernesto y a Jorge, cuyas puertas siempre estuvieron abiertas para mí y cuyas palabras de aliento y ánimo no me dejaron rendirme.

Opinión del tutor

Resumen

En este trabajo se desarrolla una solución informática al problema de acceso a la información de Recursos Humanos que presenta la Universidad de La Habana. La entidad actualmente cuenta con un sistema poco estable e insostenible por diversos factores (Directorio Único), entre los que están la pérdida de códigos fuentes y la descentralización desorganizada de los servicios que lo componen. El objetivo es sustituir este sistema por uno más robusto, estable y eficiente, que sea fácilmente modificable y sostenible. Se propone un sistema basado en el protocolo LDAP y una interfaz que permita la comunicación con el mismo de manera fácil, así como la unificación de los servicios actualmente dispersos en uno solo (o la menor cantidad posible). Para esto la propuesta se compone un servidor LDAP, una API y una interfaz web. Como culminación del trabajo se espera la adopción de este sistema en la Universidad de La Habana.

Palabras clave: Directorio Único, LDAP, API, interfaz web.

Abstract

In this project a computer solution to the problem of access to Human Resources information presented by the University of Havana is developed. The entity currently has an unstable and unsustainable system due to various factors (Directorio Único), among which are the loss of source codes and the disorganized decentralization of the services that comprise it. The objective is to replace this system with a more robust, stable and efficient one that is easily modifiable and sustainable. It is proposed a system based on the LDAP protocol and an interface that allows communication with it in an easy way, as well as the unification of the services currently dispersed into one (or as little as possible). For this, the proposal consists of an LDAP server, an API and a web interface. As the culmination of the work, the adoption of this system at the University of Havana is expected.

Keywords: Directorio Único, LDAP, API, web interface.

Índice general

Índice de figuras

Introducción

La necesidad de desarrollar los medios de comunicación y la transmisión de información es una de las características que distingue a nuestra civilización desde hace milenios. Desde que se inventó la imprenta hace aproximadamente 6 siglos, la humanidad ha generado incontables volúmenes de texto para almacenar conocimiento de todo tipo. No obstante, no es hasta la década de 1960, con la adopción y proliferación de las computadoras y el mantenimiento de registros digitales, que se alcanzaría una verdadera revolución en los medios de almacenamiento, transmisión y en la accesibilidad a la información a escala global. Esta revolución conocida como Revolución Digital o Era de la Información gira en torno a las nuevas tecnologías e Internet, siendo la catalizadora de una enorme explosión tecnológica, produciendo grandes transformaciones en la sociedad. La automatización de los procesos no solo industriales, también cotidianos, mejoró (y mejora) considerablemente la calidad, rapidez y robustez de los mismos. Además, el surgimiento de las redes de dipositivos impulsó drásticamente la facilidad de comunicación y la organización de personas y organismos.

Una de las ventajas que trae consigo el establecimiento de redes de dispositivos (como las redes locales o intranet de las empresas), es la posibilidad de descentralizar el almacenamiento de la información sin comprometer el acceso a la misma. Por ejemplo, gracias a estas redes un centro como la Universidad de La Habana, que posee varios departamentos de Recursos Humanos independientes (en las distintas facultades y demás instituciones que pertenecen a la misma), puede gestionar la información de todos estos departamentos de la universidad como un todo. Esto facilita la consulta y la modificación de esta información, que puede ser gestionada de manera adecuada con un único sistema.

Actualmente la red de la universidad cuenta con varios servicios consumidores y fuentes de diferentes tipos de información relacionada con Recursos Humanos. Como ejemplo tenemos el servicio que gestiona la conexión de los usuarios a la internet, el cual necesita consultar por ejemplo el curso actual

en caso de un estudiante. Este y otros servicios (como el correo) carecen de un sistema que administre de manera adecuada el flujo de comunicación entre los mismos.

De estos servicios se benefician no solo las facultades pertenecientes a la Universidad, también se benefician otras instituciones asociadas a la misma como el Instituto Superior de Diseño (ISDI), el Instituto Superior de Ciencia y Tecnología Aplicada (InSTEC), el IFAL, el Jardín Botánico, y la DOM.

La gran mayoría de estos requiere la apropiada verificación del usuario que solicita consultar o agregar información (sobre todo para el manejo de los diferentes roles y permisos con los que cuentan las personas encargadas de manipular los datos). Como ejemplo tenemos los dos servicios mencionados anteriormente (acceso a internet y correo), los cuales requieren de la adecuada verificación del usuario para su correcto uso, además de verificación de permisos (roles), para separar las responsabilidades en la utilización (permiso para crear listas de distribución en el correo o retirar el acceso a internet a usuarios por uso indebido).

Debido a esto y con el objetivo de centralizar este proceso de verificación se han ido acumulando sobre el dominio `directorio.uh.cu` varias actualizaciones progresivas que conforman lo que se conoce como "Directorio Único de la Universidad de La Habana".

Este sistema tiene un conjunto de responsabilidades básicas. Entre las mismas encontramos por ejemplo ofrecer información de los usuarios (tanto de trabajadores como de estudiantes), extrayéndola de las fuentes de datos correspondientes. En el caso de los trabajadores la fuente es un conjunto de bases de datos conocido como ASSETS. Estas bases de datos se obtienen de distintas maneras dependiendo de la distancia y de la caldidad de la conexión con los destinos (por ejemplo la base de datos de IFAL se trae a mano en un disco duro o memoria flash periódicamente). Por la parte de los estudiantes, la fuente es un sistema denominado SIGENU, del cual se obtiene la información consultando una serie de servicios SOAP (Simple Object Access Protocol o Protocolo Simple de Acceso a Objetos), sin la necesidad de descargar ni copiar bases de datos. Esta información de los usuarios se publica a través de una interfaz web (en el dominio de directorio), y además se ofrece una API que otros servicios (o los administradores) de la red pueden utilizar para autenticar un usuario (existe una plataforma openID a partir de los datos almacenadosen el Directorio).

Sobre este dominio web se integran y administran servicios que funcionan sobre diferentes tecnologías. Esto no es necesariamente un problema, empresas como Google poseen plataformas que funcionan sobre diversas tecnologías (como Google+ [2]) y tienden a integrar cada vez mayor can-

tidad de servicios. Aún así, en este caso en particular, el Directorio Único en su creación no fue pensado para integrarse con servicios como SQUID o POSTFIX-DOVECOT, sino que fue diseñado para brindar la información asociada a los usuarios y un módulo basado en OAuth (específicamente openID) para la autenticación. Este diseño conllevó a que a integración de nuevos servicios requiriera desarrollar capas intermedias para comunicar al Directorio con estos. Estas capas intermedias se han implementado en su mayoría como módulos independientes, en distintos lenguajes y tecnologías, y se ha agregado código al directorio para suplir las necesidades de estas capas intermedias sin seguir un estándar. Esto trae consigo dificultad a la hora de integrar nuevos servicios al directorio, y además, hace sumamente difícil darle mantenimiento a el código, ya que una modificación en el mismo puede traer como consecuencia la necesidad de modificar en cascada varios servicios independientes (que como ya mencionamos están en varios lenguages de programación y diferentes tecnologías).

Presentado este problema, el objetivo general de este proyecto es proveer a la Universidad de La Habana de un servicio que se encargue de administrar los procesos de obtención, almacenamiento y publicación de la información relacionada con Recursos Humanos, que sea robusto, modular, fácil de integrar, fácil de mantener y fácil de modificar.

Con este objetivo en mente, el primer paso hacia la implementación de la solución consiste en un análisis del problema desde un punto de vista computacional. Este análisis se sustenta en la respuesta a una serie de preguntas, donde la primera y tal vez más importante es: ¿de qué forma almacenamos toda esa información? Este objetivo se apoya en una tesis que se desarrolla en paralelo, la cual se encarga del despliegue de un servidor LDAP [6, 5] para la persistencia de la información. Esa tesis también se encarga de la forma en que esta información es almacenada (esquema de los datos).

Continuando con el análisis la siguiente pregunta puede ser: ¿qué método se debería utilizar para publicar y gestionar esta información? Actualmente existen varias implementaciones de servicios como SQUID que pueden integrarse de manera directa con un servidor LDAP para la autenticación. Aún así, el uso de LDAP no es estandarizado, y para no recurrir a uno de los problemas del actual directorio, como es la necesidad de implementar capas intermedias para comunicar los servicios, en esta tesis se propone la implementación de una API, que funcione como ese intermediario para comunicar todos los servicios que lo requieran con el servidor LDAP. Esto hace que la integración sea más fácil, y que la modificación del servidor, o de los servicios utilizados, solo conlleve la modificación mínima de esta API.

Además de esto, la implementación de una API como servicio web (utlizando HTTP como protocolo) facilita la integración de una interfaz web que sustituya la interfaz del actual directorio. Este es también un objetivo de esta tesis, desarrollar una interfaz web que facilite la gestión manual de la información (agregar externos al sistema, buscar datos de usuarios...).

Llegado a este punto podemos afirmar que la hipótesis a partir del objetivo es: la implementación de un sistema que facilite la integración del servidor LDAP de la universidad, con los distintos servicios que dependen de la información de Recursos Humanos, resolviendo la obtención de los datos a partir de las fuentes correspondientes, y brindando una interfaz web para gestionar manualmente la información que lo requiera.

Además se propone que el API brinde una bolsa de servicios para publicar información a partir del servidor LDAP (no necesariamente almacenada en el mismo, pero si dependiente), para los servicios que la requieran. Como ejemplo tenemos la cuota de internet, que se calcula a partir del curso del estudiante (almacenado en el servidor LDAP), pero no necesariamente se almacena en este servidor.

Para esto es necesario investigar sobre el estado del arte del actual Directorio, de las distintas tecnologías para implementar una API que satisfaga todas las necesidades anteriormente mencionadas (incluyendo la extracción de información de las fuentes), y de las tecnologías para la implementación de una interfaz web a partir de los servicios que brinda el API.

Capítulo 1

Directorio Único

1.1. Surgimiento

El Directorio Único de la Universidad de La Habana es un conjunto de servicios, que fueron implementados con el objetivo de acceder a los datos del personal de la Universidad de la Habana (estudiantes, trabajadores y externos). En el momento de su surgimiento, se hacia necesario disponer de un sistema que unificara las principales fuentes de datos de Recursos Humanos. De esta manera se buscaba presentar una interfaz para el manejo de esta información. El conjunto de servicios que compone el actual Directorio fue tomando forma paulatinamente al integrar soluciones aisladas a la interfaz inicial (soluciones en distintas tecnologías y lenguajes de programación). Entre los principales y más utilizados podemos destacar un mecanismo de autenticación OAuth para los sitios y servicios de la intranet (proxy, correo...). Todos los servicios implementados sobre Directorio funcionan hasta hoy. Algunos se encargan de denegar o permitir el acceso de los usuarios a determinados recursos brindados por la Universidad. Se puede tomar como ejemplo, el acceso al servicio que administra la asignación de viajes internacionales al personal de la Universidad. Otro ejemplo, un poco más palpable, lo tenemos en el servicio que decide la cuota de internet asociada a cada usuario. Todos estos servicios se basan en información procedente de Recursos Humanos (el año que cursa, en caso de ser estudiante, o el cargo y departamento, en el caso de los trabajadores).

La idea principal de un Directorio Único, es por supuesto centralizar el acceso a la información con todas las ventajas que esto brinda (estabilidad en el sistema, velocidad de acceso a la información, accesibilidad y facilidad de administración...), ya que debido a la estructura de la Universidad esta

información no se genera de manera centralizada. Existen diversas fuentes de las que se nutre el Directorio Único. Una de ellas es el Sistema de Gestión para la Nueva Universidad (SIGENU), una plataforma web con una interfaz (API y página web) para consultar y administrar información relacionada con los estudiantes usando el protocolo SOAP (Simple Object Access Protocol). Este Sistema de Gestión es administrado por las secretarias de cada una de las facultades que forman parte de la Universidad y es referente solo a los estudiantes. Otra de las fuentes es el conjunto de Bases de Datos en sqlserver de los departamentos de recursos humanos de cada una de las unidades presupuestadas de la Universidad (UH, IFAL, JBN, UPA, ISDI, INSTEC). Sobre esta última el acesso es más restringido debido a que en el proceso de informatización de esta información se utilizó un software privativo (el sistem no acepta modificaciones).

Entre los datos más relevantes que provee como servicios el Directorio Único están las credenciales de los usuarios de la red universitaria, debido a que esta información es la base del otorgamiento de permisos a la hora de utilizar servicios en la intranet. Esta información permite:

1. **Autenticar al usuario:** Comprobar que la persona que solicita un servicio es quien dice ser.
2. **Administrar el acceso:** En correspondencia del nivel de privilegio de un usuario, permitir o no el acceso a ciertos servicios.

Además se almacenan otros datos dependiendo del tipo de usuario (estudiante, trabajador o externo), como por ejemplo:

1. Año que cursa (en caso de ser estudiante)
2. Dirección Particular
3. Departamento al que pertenecen (trabajadores)
4. Datos sobre la nomina (trabajadores)
5. Puesto que ocupa (trabajadores)
6. Cargos importante (si es que los posée)

Por último a estos datos que persistidos en el Directorio se le asocia información adicional para monitorear y gestionar la actividad realizada por el usuario. Durante su estancia en la red, podemos registrar los momentos

en que se autentica, en que sistema lo hace, la cantidad de cuota de internet consumida, etc.

En el listado siguiente, se encuentra un ejemplo de una posible respuesta ofrecida por el directorio cuando se consultan los datos de un trabajador de la UH.

```
<TrabajadorInfoCuote>
<Id>15869</Id>
<CatOcupacional>técnicos docentes principal</CatOcupacional>
<Docente>Si</Docente>
<CatDocenteInvestigativa>Instructor</CatDocenteInvestigativa>
<Contrato>Indeterminado</Contrato>
<Cargo>INSTRUCTOR</Cargo>
<Adiestrado>No</Adiestrado>
<AdministradorArea>No</AdministradorArea>
<Tecnico>Si</Tecnico>
<TecnicoInformatico>No</TecnicoInformatico>
<EspecialistaPrincipal>No</EspecialistaPrincipal>
<Cuadro>No</Cuadro>
<Asset>1</Asset>
<Departamento>DIRECCION DE INFORMATIZACION</Departamento>
</TrabajadorInfoCuote>
```

1.2. Problemas del sistema actual

El sistema, tal y como existe en este momento, presenta varios problemas. Esta tesis pretende brindar una propuesta de solución para la mayoría de estos, así como su implementación. Los problemas son:

1.2.1. Mantenimiento y extensibilidad del Sistema

Debido a la naturaleza del surgimiento del Directorio Único, es decir, el acoplamiento de varios servicios de manera escalonada sobre la idea inicial (servicios implementados en varias tecnologías y lenguajes de programación), cada componente es demasiado dependiente de la forma en que las demás brindan sus correspondientes funcionalidades. Esto se debe a que la interacción entre las mismas ha sido configurada mediante un enfoque estático. El propio sistema no dispone de herramientas que permitan su modificación de una manera cómoda para los encargados de su mantenimiento. Dicho enfoque dificulta enormemente las tareas de actualización del sistema, las

cuales son necesarias para poder adecuar el mismo a las nuevas condiciones y necesidades que van surgiendo en la red a través de los años. De hecho, actualmente el personal encargado del mantenimiento de Directorio no puede responder a las necesidades de actualización. La principal causa de esta desatención, es que los desarrolladores de Directorio perdieron muchos de los cambiosen su historial de código. Tan importante es la pérdida de este historial que imposibilita la recuperación de la lógica del Directorio actual partiendo solamente del código almacenado.

1.2.2. Desuso de datos almacenados

Desde el surgimiento del Directorio Único, se han ido incorporando nuevos campos a las fuentes de información del sistema. Estos cambios han tenido como objetivo suplir las necesidades que ocupan a la Universidad en cada nuevo período escolar.

Actualmente muchos de esos campos han dejados de ser útiles para la Universidad. Como consecuencia de la rigidez del Directorio, cualquier cambio, sobre todo aquellos cuya repercusión y alcance no se conocen, podrían significar la caída del sistema por tiempo indefinido. De ahí que se siga la filosofía de que "... lo que funciona no se toca...". Pero mantener este enfoque, provoca una sobrecarga innecesaria para el sistema, que aunque pueda ser pequeña, no deja de ser significante. Dicha sobrecarga se refleja sobre todo en el espacio ocupado por la información en disco.

1.2.3. Incumplimiento de políticas de baja de usuarios

Este es otro problema en el cual se incurre con bastante frecuencia en la Universidad. Debido a la volatilidad de algunos contratos concertados con personal ajeno a nuestro centro de altos estudios.

Sucede frecuentemente que al dar de baja a estos usuarios, sus cuentas son eliminadas con efecto casi inmediato, lo cual va en contra de los protocolos usualmente implementados en estos casos. Generalmente se debe espera una cierta cantidad de días para implementar la eliminación total de las cuentas. De esta manera se puede prevenir la pérdida de acceso a servicios críticos, como son el correo, el proxy y la nube recientemente despliega en la intranet de la Universidad. Muchas veces estos servicios son desarrollados, administrados y mantenidos por agentes externos a la Universidad.

1.2.4. Carencia de protocolos comunes para servicios externos

A menudo, se implementan nuevos servicios en la red de la Universidad. Generalmente estos servicios necesitan tener control de acceso sobre los recursos que brindan a sus usuarios. Esto implica el tener que desarrollar para cada nuevo servicio, un mecanismo de autenticación de usuarios. Este mecanismo además tiene que ser capaz de brindar una funcionalidad para administrar los roles o grupos a los que pertenecen dichos usuarios. Un enfoque más útil, es el de delegar esta tarea a un sistema externo y centralizado. De esta forma se evita el tener que repetir el desarrollo de la misma funcionalidad para cada servicio.

Teniendo esto en cuenta, es que se pretende implementar una API Rest que permita modificar la lógica detrás de la información brindada, sin que esto implique modificar todos los servicios que consuman información de nuestro sistema.

1.3. Propuesta de solución

Con el objetivo de subsanar dichos problemas, pretendemos desarrollar un sistema capaz de sustituir el Directorio Único. Nuestro enfoque va orientado a desplegar un servidor que implemente el protocolo LDAP. Este protocolo define un servicio de directorio optimizado para las operaciones de búsqueda. Además posee facilidades para la organización de los datos, asociándolos a entidades, grupos y cualquier otra unidad organizacional en la que se necesiten agrupar a los datos. Dichos datos serán consumidos directamente de las mismas fuentes de las que se alimenta Directorio Único. Existirá, para esto, un capa de software intermedio capaz de transformar los datos a un formato compatible con el protocolo LDAP, específicamente el formato LDIF. Dicho formato propone la declaración de los atributos, que componen la información de un usuario, a través de pares de llaves y valores. A continuación se puede observar, como ejemplo, una entrada del servidor LDAP expresada en este formato. La misma representa la información comúnmente almacenada con respecto a un usuario que no está directamente asociado a la Universidad.

```
dn: uid=labf@fq.uh.cu,ou=Externo,dc=uh,dc=cu
area: N/D
assets: 1
cargo: JEFE DE DEPARTAMENTO (ADM FACULTAD)
```

```
categoriadocenteinvestigativa: N/D
centrodegraduacion: N/D
ci: 38081015203
ciudadania: N/D
cn: Luis Enrique
correo: labf@fq.uh.cu
cuotainternet: 0
dependencia: UH: FACULTAD DE QUIMICA
direccion: N/D
direcciondelcentro: N/D
edad: 23
esbaja: FALSE
escuadro: TRUE
fechadecreacion: 1486962000
fechadebaja: 2145889787
fechaderegistro: 1484542800
gradocientifico: N/D
lugardenacimiento: N/D
municipio: N/D
objectclass: Externo
objectclass: top
provincia: N/D
raza: Blanca
sexo: M
sn: brahin Fuente
tienechat: TRUE
tienecorreo: TRUE
tieneinternet: TRUE
uid: luis.enrique.brahin.fuente_182711
ujc: FALSE
userpassword: {SSHA}n9+lgnpEQv63Ky7smvxyISK1Gb3dq
```

En caso de ser necesario guardar alguna información que no este incluida en las fuentes, es en esta capa donde será generada.

Actualmente existe muchos servicios que saben como comunicarse directamente con el protocolo LDAP. Pero no todos incluyen esta funcionalidad. Por esta razón, es necesario implementar una interfaz, una capa de abstracción entre LDAP y el resto de los servicios, que permita su comunicación. El protocolo de comunicación más común entre servicios web es el de HTTP. Por eso pretendemos implementar una Api para garantizar la interacción

de nuestro sistema, con todos los servicios que necesiten consumir la información que almacenaremos. Los servicios que brindara dicha Api son los siguiente:

1. Consultar la información acerca de cualquiera de las usuario almacenados en el sistema.

2. Agregar la información nuevos usuarios, así como asignarles una cuota de internet y un usuarion de correo.

3. Actualizar la información de los usuarios, tanto de estudiantes, como de trabajadores y externos.

4. Definir preguntas de seguridad para cada usuario, que le permitan al mismo recuperar sus credenciales.

Realizar cualquiera de estas acciones, requiere pasar el proceso de autenticación de dicha API y poseer los permisos necesarios para la funcionalidad correspondiente.

Además como parte de la solución también implementaremos una interfaz visual para facilitar la interacción con el API.

Antes de decidir el enfoque a seguir para la implementación de la solución propuesta, se tuvo en cuenta varias tecnologías. A continuación las presentamos y argumentamos nuestra elección.

Capítulo 2

Estado del arte

2.1. LDAP

2.1.1. ¿Qué es el protocolo LDAP?

LDAP (Lightweight Directory Access Protocol o Protocolo de Acceso a Directorios Ligero) es un protocolo perteneciente a la capa de aplicaciones [1], tanto para servidores como para clientes. Es abierto y multiplataforma. Está pensado para la implementación de servicios de directorio, como son: **IBM Security Directory Server**, **Active Directory**, **Oracle Internet Directory** y **OpenLDAP**; facilitando el acceso rápido a la información almacenada. Presenta una estructura arbórea, la cual organiza la información en ramas y permite realizar búsquedas de manera eficiente, debido a que la cardinalidad de las posibles repuestas se reduce a medida que se avanza por cualquiera de estas ramas.

La selección de la implementación utilizada es objetivo de la tesis que se encarga del servidor LDAP, en esta se utiliza el este protocolo para comunicarse con este servidor, por eso se menciona qué es este protocolo. En los anexos se puede encontrar información más específica sobre el mismo, pero no es objetivo principal de esta investigación.

2.2. Docker

2.2.1. ¿Qué es Docker?

Docker es una plataforma de software libre que permite crear, desplegar y administrar contenedores de aplicaciones virtualizadas, sobre un mismo sistema operativo. Facilita la integración de distintas aplicaciones indepen-

dientemente del sistema que estas necesiten para funcionar. Hasta el momento de escritura de la presente tesis, Docker es capaz de crear contenedores basados en cualquier sistema Unix-Like [1] y en Windows. Todavía no existe forma de hacerlo basado en macOS [2].

2.2.2. Ventajas de utilizar Docker

Docker es sumamente útil en el proceso de despliegue de aplicaciones. A través de esta tecnología es posible planificar todo el proceso de despliegue de un sistema, plasmarlo en una secuencia de pasos lógicos y luego simplemente ejecutarlo. De esta manera se reduce el tiempo necesario para este proceso evitando errores frecuentes y acciones repetitivas.

En el caso específico que concierne a esta tesis, es posible utilizar Docker para automatizar todo el proceso de despliegue en producción de la solución que se propone.

2.2.3. ¿Quién utiliza Docker [3]?

Muchas grandes empresas se han apoyado en Docker para automatizar el desarrollo de software. Como ejemplo encontramos algunas muy conocidas mundialmente como son Spotify, Uber, eBay o PayPal.

Tomando como caso específico a eBay, ésta ha centrado sus esfuerzos en la implementación de Docker en su proceso de integración continua, estandarizando sus instalaciones mediante despliegues en una red distribuida de servidores que se ejecutan y trabajan en grupo. Esto permite a los desarrolladores o testers (encargados de realizar pruebas de software) sólo tengan que montar contenedores sobre sus sistemas para poder ponerse a trabajar, con toda la configuración, herramientas o servicios precargados, lo que agiliza enormemente los tiempos de desarrollo o testeo de versiones.

En general estas empresas (y muchas otras) vieron en Docker la respuesta a sus preguntas de cómo poder obtener mayor rendimiento invirtiendo menos horas de supervisión y desarrollo. Creando una imagen que contenga el sistema, las configuraciones y servicios listos para ser desplegados; se ahorraron multitud de horas de gestión de sistemas, reparación, recuperación y reinstalación de éstos.

Gracias a los contenedores las pruebas se realizan en entornos "cerrados" por lo que si se diese algún error la mayor complicación que se encontrarán será la de volver a desplegar el contenedor de nuevo.

[1] Sistemas que utilizan el kernel de Linux

[2] `https://www.apple.com/la/macos/what-is/`

2.2.4. ¿Qué es un contenedor de docker?

Como parte fundamental de Docker se encuentran los contenedores. Estos, según los propios desarrolladores, *son unidades de software estandarizadas* [3]. Se encargan de empaquetar el código y las dependencias de la aplicación de manera que pueda ser ejecutado en cualquier entorno capaz de ejecutar Docker. Para ejecutar un contenedor, se necesita una imagen de Docker base. Una imagen de Docker no es más que un paquete que contiene todas las dependencias y configuraciones necesarias para ejecutar un contenedor a partir de esta. Los contenedores aíslan al software del entorno en que son ejecutados y aseguran el correcto funcionamiento independientemente de las posibles diferencias entre un entorno u otro. Sin embargo, dicho aislamiento no impide la comunicación entre varios contenedores. Existen mecanismos que permiten interactuar a través de servicios de red. Incluso es posible crear una red privada que incluya ciertos contenedores. Con esta facilidad, se puede diseñar soluciones estructuradas en módulos atómicos que se integren como sistema.

2.2.5. Docker Compose

Compose es una herramienta para definir y ejecutar aplicaciones en Docker con contenedores múltiples. Con Compose, se utiliza un archivo YAML para configurar los servicios de la aplicación. Luego, con un solo comando, crea e inicia todos los servicios desde la configuración [10]. Un ejemplo de archivo de configuración para Docker Compose es el siguiente:

```
version: '3'
services:
    memcached:
        build:
        context: .
        dockerfile: ./memcached-dockerfile
        image: memcached
        container_name: memcached_server
        expose:
        - 11211
        ports:
        - "11211:11211"
        restart: always
```

[3] `https://www.docker.com/resources/what-container`

```
        command: memcached -u root
db :
    image: "mcr.microsoft.com/mssql/server"
    container_name: sqlserver_db
    environment:
    SA_PASSWORD: "P@ssw0rd"
    ACCEPT_EULA: "Y"
api :
     build: .
     image: flask_restful
     container_name: ldap_api
     expose:
     - 5000
     ports:
     - "5000:5000"
     volumes:
     - .:/api
     restart: always
     working_dir: /api
     command: python run.py
     depends_on:
     - memcached
     - db
ui :
   build:
   context: .
   dockerfile: ./ui-dockerfile
   image: react_ui
   container_name: ldap_ui
   ports:
   - "3000:3000"
   working_dir: /ui
   volumes:
   - ./UI:/ui
   restart: always
   command: bash -c "npm install && npm start"
networks:
      default:
           external:
                name: tesis
```

2.2.6. Casos de uso comunes de Docker Compose

Entornos de desarrollo

Cuando se desarrolla software, la capacidad de ejecutar una aplicación en un entorno aislado e interactuar con ella es crucial. La herramienta Compose se puede utilizar para crear el entorno e interactuar con él.

El archivo de configuración proporciona una forma de documentar y configurar todas las dependencias de la aplicación (bases de datos, colas, cachés, API de servicios web, etc.). Con esta herramienta, se puede crear e iniciar uno o más contenedores para cada dependencia con un solo comando (docker-compose up).

Juntas, estas características proporcionan una manera conveniente para que los desarrolladores comiencen un proyecto. Compose puede reducir una "guía de inicio para desarrolladores"de varias páginas a un solo archivo Compose y algunos comandos.

Entornos de pruebas automatizadas

Una parte importante de cualquier proceso de implementación continua o integración continua es el conjunto de pruebas automatizadas. Las pruebas automatizadas de extremo a extremo requieren un entorno en el que ejecutarse. Compose proporciona una forma conveniente de crear y destruir entornos de prueba aislados. Al definir el entorno completo en un archivo de configuración, se puede crear y destruir estos entornos con solo unos pocos comandos:

docker-compose up -d
./run_tests
docker-compose down

2.3. API

2.3.1. ¿Qué es una API?

Una API es un conjunto de funciones y procedimientos que cumplen una o muchas funciones con el fin de ser utilizadas por otro software. Las siglas API vienen del inglés Application Programming Interface. En español

sería Interfaz de Programación de Aplicaciones. Es una especificación formal sobre cómo un módulo de un software se comunica o interactúa con otro. [4]

2.3.2. Ventajas de implementar una API

Las API simplifican en gran medida el trabajo de un creador de programas, ya que no tiene que escribir códigos desde cero. Estas permiten al programador utilizar funciones predefinidas para interactuar con el sistema operativo o con otro programa. También reducen los costes de mantenimiento, al organizar de forma estructurada los sistemas y procesos internos, lo cual permite la integración de nuevos proyectos de una manera uniforme. También agilizan los procesos de transformación, ya que mientras la interfaz sea la misma, los procesos internos pueden cambiar tanto como sea necesario, sin la necesidad de modificar los servicios que consumen una API específica.

En el caso de la solución que propone esta tesis, tener una API como intermediaria entre el servidor LDAP, las demás fuentes de información (SIGENU y Bases de Datos de trabajadores) y los mecanismos de gestión de información (interfaces visuales para manejo de externos, cuotas, etc...), permite manipular los diversos componentes del Directorio Único utilizando un lenguaje común muy estandarizado (protocolo HTTP en este caso). Esto permite a los distintos clientes consumidores de la información de directorio abstraerse del protocolo LDAP (menos estandarizado). Además permite que al modificarse la estructura del Directorio, los clientes no tengan que ser modificados, ya que al modificar solamente la API como capa intermedia podemos cambiar la forma de obtener información de las fuentes mientras mantenemos la estructura a la hora de publicar la información.

2.3.3. Tecnologías para implementación más utilizadas

Para la implementación de una API como servicio web existen gran cantidad de *frameworks* [5] sobre una gran variedad de lenguajes de programación. Entre los más utilizados tenemos:

ASP.NET Core

ASP.NET Core es un framework web de código abierto implementado por Microsoft, para desarrollar aplicaciones web dinámicas utlizando C Sharp o Visual Basic como lenguaje de programación.

[4] `https://hipertextual.com/archivo/2014/05/que-es-api/`

[5] `https://www.ecured.cu/Framework`

Este framework es un rediseño de ASP.NET 4.x, para crear aplicaciones web. Aquí tenemos los beneficios que brinda según la documentación oficial de Microsoft [4]:

- Una historia unificada para desarrollar interfaces web y APIs.
- Arquitectura diseñada para pruebas.
- Las páginas Razor facilitan y hacen más productiva la codificación enfocada en páginas web.
- Blazor te permite utilizar C Sharp junto a Javascript en el navegador.
- Permite desarrollar y ejecutar aplicaciones tanto en Windows, como en Linux y macOS.
- Es de código abierto y basado en la comunidad.
- Permite una fácil integración de frameworks del lado del cliente modernos.
- Se adapta fácilmente a distintos flujos de desarrollo modernos.
- Aplicaciones listas para desplegar en la nube.
- Inyección de dependencias integrada.
- Un flujo de peticiones HTTP ligero, eficiente y modular.
- Permite ser desplegado utilizando IIS, Nginx, Apache, Docker.
- Posee varias herramientas que simplifican el desarrollo web moderno.

ASP.NET MVC

Este es otro framework desarrolladores por Microsoft basado en .NET, con la diferencia fundamental de que implementa el patrón Modelo Vista Controlador. Es de código abierto también, exceptuando ASP.NET Web Forms que es un software propietario.

El patrón Modelo Vista Controlador (MVC) es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos: el Modelo que contiene una representación de los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia; la Vista, o interfaz de usuario, que compone

la información que se envía al cliente y los mecanismos interacción con éste; el Controlador, que actúa como intermediario entre el Modelo y la Vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno [9].

Entre las ventajas de la utilización de este framework están:

- Posibilidad de dividir la lógica de negocio del diseño, haciendo el proyecto más escalable.
- Facilita el uso de URL amigables, importantes para el SEO (posicionamiento web), la mayoría de frameworks MVC lo controlan.
- Puedes utilizar abstracción de datos, como lo hacen también otros frameworks como Ruby on Rails, mediante el uso de ORMs (Object-Relational Mapping), permitiendo comunicarte con la base de datos de la aplicación de una forma orientada a objetos y no mediante SQL.

Ruby on Rails

Este es un framework web de código abierto con una de las comunidades de desarrolladores más grandes. Esta implementado sobre el lenguage Ruby. Utiliza también el patrón Modelo Vista Controlador (MVC), con las ventajas que vimos anteriormente. Es un sistema web integrado que incorpora por defecto muchas de las herramientas necesarias para escalar un sitio web de manera rápida y eficiente. Como Ruby es un lenguage dinámico [7] el desarrollo es más veloz debido a que no se necesita recompilar el código para probar cambios. Entre las principales ventajas de elegir Ruby on Rails tenemos:

- Utiliza el patrón MVC.
- Permite desplegar de forma separada contenido estático y dinámico.
- Dispone de un sistema de caché integrada que se puede utilizar a nivel de página, o en cualquier fragmento según sea necesario en cada caso.
- ORM Integrado. El famoso ORM ActiveRecord permite lanzar consultas a la DB de forma extremadamente fácil y sin necesidad de escribir nada de SQL.
- Sistema de testing integrado. Cuando hay la necesidad de implementar algoritmos complejos es mejor utilizar TDD para tener una buena cobertura que les dé validez y estabilidad.

- Ecosistema de librerías maduro. Existen librerías para casi cualquier necesidad (existe una gran cantidad de módulos denominados gemas, que agregan casi cualquier funcionalidad a la aplicación).

Django

Django es un framework gratuito y de código abierto basado en Python, que sigue el patrón Modelo Plantilla Vista. Lo mantiene Django Software Foundation, una organización independiente establecida como una organización sin fines de lucro. El objetivo principal de Django es facilitar la creación de sitios web complejos basados en bases de datos.

En este patrón MTV, la M significa Model (Modelo), la capa de acceso a la base de datos. Esta capa contiene toda la información sobre los datos: cómo acceder a estos, cómo validarlos, cuál es el comportamiento que tiene, y las relaciones entre los datos. T significa Template (Plantilla), la capa de presentación. Esta capa contiene las decisiones relacionadas a la presentación: como algunas cosas son mostradas sobre una página web o otro tipo de documento. V significa View (Vista), la capa de la lógica de negocios. Esta capa contiene la lógica que accede al modelo y la delega a la plantilla apropiada: se puede pensar en esto como un puente entre el modelos y las plantillas [8].

Es un patrón muy similar a MVC, excepto que la C (Controlador) es gestionada en el mismo framework (a diferencia de otros como ASP.NET MVC o Ruby on Rails). En Django, las vistas tienen más relación con los controladores de MVC, y las plantillas con las vistas. Resumiendo las ventajas de usar Django encontramos:

- Utiliza el patrón MTV (muy similar a MVC).
- Está escrito en Python, lenguaje con una gran cantidad de librerías para resolver diversos tipos de problemas.
- Es el único framework que por defecto viene con un sistema de administración activo, listo para ser utilizado sin ningún tipo de configuración.
- ORM Integrado.

Flask

Flask es un microframework web escrito en Python. Se clasifica como un microframework debido a que no requiere de herramientas ni librerías particulares. No tiene una capa de abstracción de base de datos ni sigue un

patrón de desarrollo específico. Las funciones específicas como la autenticación y el manejo de base de datos son brindadas por software de terceros. Entre las principales ventajas de este framework minimalista tenemos que es muy eficiente, ya que te permite seleccionar que componentes utilizar específicos a la plataforma que estés desarrollando. Es muy utilizada para desarrollar APIs debido a la facilidad con que se pueden desarrollar las mismas (muy pocas líneas de código), y debido a que está montado sobre python permite utilizar todas las libreríascon las que cuenta este poderoso lenguaje de programación. Tiene una comunidad de desarrolladores activa y una buena cantidad de módulos para agregar distintas funcionalidades a los servicios web.

2.4. python-ldap

2.4.1. ¿Qué es python-ldap?

python-ldap es una librería del lenguaje de programación Python, la cual se encarga de proveer una API orientada a objetos para acceder a servidores de directorios basados en el protocolo LDAP desde los programas de Python.

2.5. Interfaz web

Por último, existen diferentes tecnologías para implementar la interfaz web, que facilitará la interacción de los usuarios, tanto familiarizados con la programación como ajenos a la misma (secretarias y personal de Recursos Humanos), con el API e indirectamente con el servidor LDAP y las fuentes de información. Como para mantener modularizada la solución, el API y la interfaz deben ser dos programas separados, en este caso es muy factible utilizar una SPA (Single Page Application), que funcione como una aplicación independiente. Algunas de las tecnologías mas usadas para desarrollar SPAs actualmente son:

Angular

Este es un framework monolítico de código abierto desarrollado por Google, muy poderoso por la cantidad de funcionalidades internas que provee pero con una curva de aprendizaje excesiva.

React

Este es la alternativa de Facebook, ampliamente utilizada al igual que Angular, también es de código abierto y aunque no es un framework en sí (es una librería para renderizar vistas), unida con otras tecnologías como redux, se convierte en una herramienta poderosa para dessarrollar SPAs. Es muy eficiente y su curva de aprendizaje es más suave que la de Angular.

Vuejs

Este es la alternativa más joven de las tres, la cuál aún con la comunidad más pequeña a obtenido una gran cantidad de estrellas en github.com debido a que tiene la curva de aprendizaje más suave, y a las comodidades que brinda por su semántica sensilla a los desarrolladores.

Capítulo 3

Implementación del sistema

3.1. Consideraciones previas

La implementación del servidor LDAP es parte del desarrollo otra tesis que complementa a esta en la creación de la solución final. En esta se implementó lo concerniente al API y la interfaz web, incluyendo la generación de los archivos LDIF utilizados para popular y modificar de manera periódica el servidor LDAP mediante dos módulos de Python independientes al API (aunque esta los utiliza); uno para la extracción de la información pertinente a los estudiantes y otro para la pertinente a los trabajadores. Además el API brinda una bolsa de servicios a los que se pueden asociar otros sistemas que necesitan consumir la información almacenada en el servidor (como por ejemplo la cuota de internet de un estudiante).

3.2. Extracción de la información de estudiantes y trabajadores

Como mencionamos antes, de la extracción de la información se encargan dos módulos independientes desarrollados en Python, los cuáles consumen la información de Recursos Humanos de los destinos correspondientes y generan archivos LDIF que utiliza el servidor LDAP para modificar los datos almacenados. Estos módulos son:

3.2.1. Módulo para generar los LDIF de los estudiantes

Este módulo se llama sigenu-client. Consume utilizando el protocolo SOAP los recursos que ofrece el sistema SIGENU para consultar información

relacionada a los estudiantes de la universidad. Específicamente consume dos recursos: Faculties y getStudentsByFaculty. El primero se utiliza para obtener todas las facultades miembros de la universidad, y a partir de estas se utiliza el segundo para obtener todos los estudiantes matriculados en cada una de estas.

Al desarrollar este módulo se tuvo que lidiar con problemas inherentes al sistema SIGENU, como datos en mal estado (caracteres no alfanuméricos) y duplicación de cuentas. La política para el primer caso es no almacenar los estudiantes con datos en mal estado en el servidor LDAP, para que soliciten la apertura de la cuenta nuevamente y quede en correcto estado. La política para el segundo caso es no almacenar ninguna de las cuentas para que también sea necesario la creación de una nueva en correcto estado.

La generación del archivo LDIF final queda estructurada de la forma que puede apreciarse en el siguiente ejemplo:

1. **becado :** Externo
2. **carrera:** Ciencias Alimentarias
3. **ci:** 00071068259
4. **ciudadania**[1]**:** Cuba
5. **cn:** Jeidis Elisa
6. **correo:** None
7. **edad:** 19
8. **facultad:** None
9. **grado:** 1
10. **grupo:** 2
11. **lugardenacimiento:** None
12. **municipio:** Playa
13. **pais:** Cuba
14. **provincia:** La Habana

[1]sin acento por la codificación del archivo generado, para facilitar la integración con el servidor LDAP

15. **raza:** B
16. **sexo:** Femenino
17. **tipodecurso:** Curso Regular Diurno
18. **idsigenu:** 2b2f70cf:1659152c466:603
19. **dn:** Estudiante
20. **idfacultad:** 223.0.06816_12
21. **cuotadeinternet:** 0
22. **pcc:** False
23. **ujc:** False
24. **esBaja:** True
25. **objectclass:** Estudiante
26. **objectclass:** posixAccount
27. **objectclass:** shadowAccount
28. **uidNumber:** 5000
29. **gidNumber:** 10000
30. **homeDirectory:** /
31. **uid:** 5000

3.2.2. Módulo para generar los LDIF de los trabajadores

Este módulo se llama ldif-from-database. Se encarga de obtener la base de datos de Recursos Humanos (Assets), extraer la información pertinente a los trabajadores, y generar el archivo LDIF utilizado por el servidor LDAP.

La generación del archivo LDIF final queda estructurada de la forma que puede apreciarse en el siguiente ejemplo:

1. **ci:** 00010868492
2. **cn:** DAIMIS DE LA CARIDAD
3. **area:** COMUNICACION

4. **cargo:** SECRETARIA
5. **sexo:** F
6. **sn:** HECHEVARRIA GUERRA
7. **dn:** 2
8. **objectclass:** Trabajador
9. **objectclass:** posixAccount
10. **objectclass:** shadowAccount
11. **uidNumber:** 5000
12. **gidNumber:** 10000
13. **homeDirectory:** /
14. **uid:** 5000

3.3. Implementación de la API

La API se desarrolló utlizando el framework Flask de Python, debido a la eficiencia de este framework minimalista, y al poder de este lenguaje de programación en cuanto a soluciones ya implementadas (librerías). Consiste en un conjunto de recursos creados utilizando la librería flask-restful a los cuales se accede utilizando el protocolo HTTP (los recursos se publican como URLs). Utilizando la librería python-ldap, el API se conecta y se comunica con el servidor LDAP.

3.3.1. Listado de recursos

/login

Este recurso es utilizado para autenticar a los usuarios en el API.
Método: POST
Argumentos: [username, password]
Objeto respuesta: []

/logout

Este recurso es para terminar la sesión de un usuario autenticado en el API.

Método: POST
Argumentos: []
Objeto respuesta: []

/usuarios

Este recurso es para obtener la información pertinente un conjunto de personas pertenecientes a la universidad en base a parámetros de búsqueda.

Método: GET
Argumentos: [filtros]
Objeto respuesta: [usuarios: [...]]

/trabajadores

Este recurso es accesible mediante 3 métodos HTTP, para realizar 3 acciones distintas. Método GET para obtener la información pertinente a un conjunto de trabajadores pertenecientes a la universidad en base a parámetros de búsqueda. Método POST para agregar un trabajador al directorio a partir de su Carnet de Identidad. Método PATCH para actualizar los trabajadores en el directorio a partir de las Bases de Datos de Recursos Humanos (cuando se hace mención al directorio nos referimos al servidor LDAP que almacena la información).

Métodos: GET, POST, PATCH
Argumentos: GET: [filtros], POST: [ci], PATCH: []
Objeto respuesta: GET: ['workers': [...]], POST: [], PATCH: []

/estudiantes

Este recurso es accesible también mediante 3 métodos HTTP, para realizar 3 acciones distintas. Método GET para obtener la información pertinente a un conjunto de estudiantes de la universidad en base a parámetros de búsqueda. Método POST para agregar un estudiante al directorio a partir de su Carnet de Identidad. Método PATCH para actualizar los estudiantes en el directorio a partir de SIGENU (cuando se hace mención al directorio nos referimos al servidor LDAP que almacena la información).

Métodos: GET, POST, PATCH
Argumentos: GET: [filtros], POST: [ci], PATCH: []

Objeto respuesta: GET: ['students': [...]], POST: [], PATCH: []

/externos

Este recurso es accesible mediante 2 métodos HTTP, para realizar 2 acciones distintas. Método GET para obtener la información pertinente a un conjunto de externos de la universidad en base a parámetros de búsqueda. Método POST para agregar un externo (persona que utiliza de forma temporal los servicios de la universidad, como un profesor de otra universidad que viene a impartir un curso de verano, etc...) al directorio (cuando se hace mención al directorio nos referimos al servidor LDAP que almacena la información).

Métodos: GET, POST
Argumentos: GET: [filtros], POST: [nombre, ci, cuota, ...]
Objeto respuesta: GET: ['externs': [...]], POST: []

/p/preguntasdeseguridad

/p/cambiar

3.4. Interfaz web

La interfaz web se implementó como una aplicación independiente utilizando React como tecnología. Brinda las funcionalidades esenciales que actualmente presenta la interfaz del actual directorio como: la recuperación de contraseñas a partir de las preguntas de seguridad de un usuario; el cambio de contraseñas; la creación de cuentas tanto de trabajadores, de estudiantes, como de externos.

3.5. Bolsa de servicios

La bolsa de servicios no es más que una extensión de las funcionalidades básicas del API, para satisfacer las necesidades de los sistemas que necesitan consumir información que se obtiene a partir de los datos almacenados en el servidor LDAP, pero que no necesariamente están almacenados en el mismo, como la cuota de internet.

Por convención la url con la que se publican estos servicios contiene el prefijo /servicios/", por ejemplo, para obtener la cuota de internet de un estudiante se utiliza la url /servicios/cuotadeinternet/estudiantes", la cual recibe el correo del estudiante como parámetro.

Capítulo 4

Anexos

4.1. Conceptos Importantes dentro de LDAP

4.1.1. Servidor de Directorio

Un servidor de directorio, no es más que un tipo de base de datos pensada para ser utilizada directamente en la red. A diferencia de las bases de datos tradicionales[1], que representan los datos en tablas y cada instancia es una fila, en este cada entrada en el directorio es un árbol de entradas, donde cada árbol puede contener datos o ser una hoja (un árbol vacío)

4.1.2. Entradas

Cada entrada en un servidor de directorio representa una colección de información referente a cierta entidad. Está compuesta principalmente, por un Nombre Distinguido, que es el identificador unívoco de la misma. Además cuenta con un conjunto de atributos y de clases de objetos los cuales definen la estructura y el comportamiento de la entrada.

4.1.3. Nombre Distinguido (DN)

Este es el identificador unívoco de la entrada. Esta compuesto por lo que se conoce en la literatura como 'Nombres distinguidos relativos' o 'RDN' por sus siglas en inglés. Estos RDN no son más que un conjunto ordenado de pares atributo-valor. Usualmente se escogen los atributos más representativos de cada entrada para la representación del DN.

[1] Bases de datos relacionales, definidas por Edgar F. Codd en su artículo: A Relational Model of Data for Large Shared Data Banks

4.1.4. Artibutos

Los atributos son los encargados de guardar la información de cada entrada y tiene asociados un tipo y un conjunto de opciones.

Los atributos representan una parte importante del esquema del directorio LDAP. A través de estos podemos definir nuevas clases de objetos para poder suplir las necesidad de almacenamiento de información. Para poder definir tanto atributos como clases de objetos es necesario proveerle a ambos un identificador, el cual presenta un formato similar al siguiente:

`1.3.6.1.4.1.<Identificador global>.1.5`

A continuación podemos ver un ejemplo de como se puede definir un nuevo atributo:

```
1  dn: cn=UHAccount,cn=schema,cn=config
2  objectClass: olcSchemaConfig
3  cn: UHAccount
4  olcAttributeTypes: ( 1.3.6.1.4.1.53027.1.1 NAME 'assets'
5  DESC 'assets'
6  EQUALITY integerMatch
7  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 )
```

Este listado presenta el formato de un archivo ldif, con los cuales se administra tanto las configuraciones como los datos almacenados en el servidor LDAP. De la línea de código 1 a la 3, se configura bajo cuál entrada del directorio LDAP se agrega la configuración correspondiente. Luego se describe la inclusión de un nuevo atributo llamado «assets».

El identificador global, como se puede ver en el listado, no es más que un número de serie que distingue a la implementación del protocolo LDAP utilizada a nivel global. Este se puede obtener realizando una solicitud a IANA(Internet Assigned Numbers Authority) [2].

4.1.5. Clases de Objetos

Estos también representan una parte importante del esquema del protocolo LDAP. No son más que conjunto de atributos que definen la información almacenada en cada entrada. Pueden ser de dos tipos: estructurales o auxiliares. Cada entrada puede tener asociada una clase de objetos estructural y cero o más clases auxiliares. En el listado siguiente tenemos un ejemplo de como definir una nueva clase de objeto.

```
1  olcObjectClasses: ( 1.3.6.1.4.1.53027.2.1 NAME '
       UHAccount'
```

[2] `https://www.iana.org`

```
2  DESC 'Base user account for UH's authentication system'
3  SUP person
4  MUST (userPassword $ email )
5  MAY (givenName $ NoCI $ assets $ isAdmin ))
```

4.1.6. Filtros

Los filtros representan el mecanismo utilizado para realizar consultas al directorio. La lógica utilizada para filtrar las entidades almacenadas en el servidor se define a través de reglas de comparación, las cuales, a su vez se definen en los atributos.

4.2. Implementaciones más utilizadas

4.2.1. IBM Security Directory Server

Este servicio implementa las especificaciones de Internet Engineering Task Force (IETF) LDAP V3. Permite la comunicación con clientes basados en IETF LDAP V3. Esta alternativa presenta una amplia variedad de funcionalidades que facilitarían la integración con el sistema de la Universidad , pero esta herramienta es de pago, por lo que no podemos utilizarla [3].

4.2.2. Active Directory

Esta es la implementación que brinda Microsoft[4] del protocolo LDAP. Entre las principales ventajas que presenta esta implementación, se encuentra la integración que brinda con los propios programas de Microsoft como por el ejemplo el Outlook. En este caso no es necesario proveer las credenciales para utilizar estos servicios. Pero peca de los mismo que la mayoría de los programas de Microsoft. Es dependiente del sistema operativo por lo que necesita del sistema Windows Server. Para poder utilizar este sistema, es necesario comprar la correspondiente licencia y posiblemente mejores servidores que sean capaz de soportarlo.

[3] `https://www.ibm.com/support/knowledgecenter/en/SSVJJU_6.3.1/com.ibm.IBMDS.doc_6.3.1/admin_gd13.htm`

[4] `https://support.microsoft.com/es-es/help/196464`

4.2.3. Oracle Internet Directory

Oracle Internet Directory es un servicio de directorio de proposito general que facilita realizar consultas rápidas y administración centralizada de la información almacenada sobre los usuarios que utilizan la red. El mismo combina el protocolo LDAP en su version número 3 con el eficiente funcionamiento, escalabilidad y robustez de una base de datos de Oracle [5].

4.2.4. OpenLDAP

OpenLDAP es la alternativa de software libre que implementa el protocolo LDAP. Como la mayoría de las implementaciones de servicios asociados a la ideología del software libre, esta variante se ejecuta sobre sistemas basados en el kernel de Linux [6]. Esto facilita su despliegue en el ecosistema de la red universitaria, ya que la mayoría de los servidores, ejecutan sistemas basados en dicho kernel. Además esta implementación se encuentra disponible en los repositorios de las distribuciones más populares entre la comunidad de software libre, lo cual facilita la elección de uno u otro sistema base en dependencia de las necesidades.

Junto al programa de instalación, se encuentran predefinidas varios tipos de atributos y clases de objetos. Estos pretenden suplir las necesidades más comunes de aquellos que necesitan utilizar un servicio de directorio. Podemos destacar:

1. Asignar grupos a los usuarios registrados y de esta manera controlar su rango de acceso.

2. Estructurar la información almacenada de manera que simule las áreas y departamentos que componen a la Universidad.

3. Definir, sin mucho esfuerzo, aquellos atributos que se suelen almacenar sobre una persona como recurso humano o como internauta o directivo, etc.

4. Facilitar la integración con distintos sistemas de autenticación a través de las clases predefinidas.

Además de incluir estos esquemas por defecto, es fácilmente extensible. Brinda, dentro de sus funcionalidades, la capacidad de definir nuevos tipos de

[5] `https://docs.oracle.com/cd/B14099_19/idmanage.1012/b14082/intro.htm#i1001669`

[6] `https://es.wikipedia.org/wiki/Núcleo_Linux`

atributos y de clases de objetos. Incluso es posible extender los ya existentes a través de mecanismos simples de herencia de clases, así como definir campos obligatorios u opcionales.

Con respecto al apartado de seguridad, OpenLDAP permite la encriptación de la información almacenada a través de distintos métodos. Para algunos brinda soporte de manera nativa, para otros realiza el proceso de encriptación a través de la libreria CRYPT [7].

El soporte nativo lo brinda para:

1. **MD5**[8] **:** Codificación basada en el algoritmo MD5
2. **SMD5:** Codificación basada en el algoritmo MD5 con un secuencia aleatoria de caracteres conocida como salt
3. **SHA**[9]**:** Codificación basada en el algoritmo SHA-1
4. **SSHA:** Codificación basada en el algoritmo SHA-1 con un secuencia aleatoria de caracteres conocida como salt

A través de CRYPT y el formato PHC string [10] brinda soporte para:

1. **MD5:** Codificación basada en el algoritmo MD5
2. **Blowfish / bcrypt:**
3. **NTHASH:**
4. **SHA-256:**
5. **SHA-512:**
6. **Solaris MD5:**
7. **PBKDF1 with SHA-1:**

Existe además, una interfaz web ya implementada conocida phpLDAPadmin [11] que permite administrar de manera básica, el contenido del servidor LDAP. La misma permite listar los datos almacenados, modificarlos, añadir nuevas entradas, etc. En resumen, permite realizar las operaciones usuales sobre un conjunto de datos.

[7] `https://ftp.gnu.org/old-gnu/Manuals/glibc-2.2.3/html_node/libc_650.html`
[8] `https://www.ietf.org/rfc/rfc1321.txt`
[9] `https://www.ietf.org/rfc/rfc3174.txt`
[10] `https://github.com/P-H-C/phc-string-format`
[11] `http://phpldapadmin.sourceforge.net/wiki/index.php/Main_Page`

4.3. Modos de empleo usuales

4.3.1. DNS

LDAP es usualmente utilizado con una estructura de DNS. Las clases de objetos que existen por defecto en el esquema de OpenLDAP, permite simular una estructura de delegación de zonas, arborea, similar a la del DNS. Esto da la oportunidad de brindar las mismas funcionalidades de servicio de nombres de dominios y a la vez utilizar las ventajas de búsqueda y modificación de los LDAP.

4.3.2. Sistema de Autenticación

Esta implementación también brinda ventajas a la hora de implementar un sistema de autenticación de usuarios. Esto se debe principalmente al amplio soporte que tiene el protocolo LDAP para varios servicios. La posibilidad de agrupar a los usuario mediante unidades organizativas (Organizational Unit [OU]) y de representar su pertenencia a determinados grupos, permite administrar fácilmente el acceso que cada uno de los usuarios debe tener a los servicios ofrecidos por la universidad. Este modo de organizar la información de los usuarios se asemeja bastante a la manera en que se asigna permisos a un usuario en los sistemas operativos basados en Linux. De hecho, una de las funcionalidades implementadas como cliente de este protocolo, permite autenticar un usuario en una máquina, ya sea virtual o física, a pesar de que realmente no exista en el sistema. Basta con que el usuario exista en el servidor LDAP. En el caso específico de OpenLDAP, cuenta con dos clases de objetos que permiten este comportamiento. Se trata de la clase posixAccount y shadowAccount. Entre ellas guardan información referente a los atributos de un usuario en un sistema basado en linux. Las principales son: el directorio "home"del usuario, el número que lo identifica en el sistema, el grupo al que pertenece.

Recomendaciones

Conclusiones

En la presente tesis se desarrolló un sistema que da solución al problema que presenta la utilización del actual Directorio Único de la Universidad de La Habana, manteniendo las ventajas que el mismo brinda para tener acceso a la información relacionada con Recursos Humanos, y eliminando las carencias más importantes del mismo como la falta de estabilidad, de modularidad y la dificultad para actualizaciones. Para esto se analizaron distintas formas de almacenamiento de información, y mecanismos de consulta, para elegir la solución más adecuada para las especificidades del problema en cuestión.

La estrategia de solución diseñada consiste en un servidor LDAP, una API para comunicar este servidor con los servicios que lo requieran, dos módulos independientes para extraer la información de las fuentes (SIGENU y bases de datos de Assets), y una interfaz web para gestionar las funcionalidades principales (que actualmente brinda la interfaz publicada en directorio.uh.cu).

Esta solución ha sido implementada de manera que sea fácilmente despegable y modificable. Gracias al uso de la tecnología Docker el despliegue requiere una configuración manual mínima, así como ninguna dependencia en la computadora final en la que se publique el sistema como servicio (exceptuando Docker). Además, el uso de lenguajes dinámicos (como lo son Python y Javascript), disminuye el riesgo de la pérdida del código fuente, ya que la aplicación en producción no es un archivo binario ilegible, sino el código en si.

Bibliografía

[1] *TANENBAUM, ANDREW S. Redes de computadoras PEARSON EDUCACIÓN, México, 2003 ISBN: 970-26-0162-2.* (Citado en la página 12).

[2] *Top 4 open source LDAP implementations, opensource.com*, Julio 2011. `https://www.genbeta.com/desarrollo/un-vistazo-a-las-tecnologias-que-usa-google-por-uno-de-sus-responsables`. (Citado en la página 2).

[3] *3 casos reales de uso de Docker en grandes empresas*, Diciembre 2015. `https://docs.docker.com/compose/`. (Citado en las páginas VII y 13).

[4] *Introduction to ASP.NET Core*, Junio 2019. `https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-2.2`. (Citado en la página 18).

[5] *LDAP Concepts & Overview, zytrax.com*, Marzo 2019. `http://www.zytrax.com/books/ldap/ch2/`. (Citado en la página 3).

[6] *LDAP Official Site*, Junio 2019. `https://ldap.com/why-choose-ldap/`. (Citado en la página 3).

[7] *Lenguaje de programación dinámico*, Agosto 2019. `https://developer.mozilla.org/es/docs/Glossary/Dynamic_programming_language`. (Citado en la página 19).

[8] *Modelo Plantilla Vista (MTV)*, Agosto 2019. `https://www.escuelapython.com/django-1-introduccion-patron-mtv/`. (Citado en la página 20).

[9] *Modelo vista controlador (MVC)*, Agosto 2019. `https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html`. (Citado en la página 19).

[10] *Overview of Docker Compose*, Agosto 2019. `https://docs.docker.com/compose/`. (Citado en la página 14).