ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DESARROLLO DE UN PROTOTIPO DE APLICACIÓN DISTRIBUIDA PARA GESTIÓN DE CONTENIDOS BASADO EN ALFRESCO PARA LA EPN

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y REDES DE INFORMACIÓN

JOSUE DAVID BURBANO VALLEJO

DIRECTOR: ING. RAÚL DAVID MEJÍA NAVARRETE, MSc.

Quito, Febrero 2019

AVAL

Certifico que el presente trabajo fue desarrollado por Josue David Burbano Vallejo, bajo mi supervisión.

ING. RAÚL DAVID MEJÍA NAVARRETE, MSc
DIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Yo Josue David Burbano Vallejo declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

JOSUE DAVID BURBANO VALLEJO

AGRADECIMIENTO

A mis padres quienes son el soporte de mi vida; a mi mamá Lucía por tanto amor incondicional, por tanta preocupación, por su sonrisa diaria y su espiritualidad; a mi papá Ramiro por su confianza, su paciencia y su manera de hacer las cosas en excelencia que ha inspirado mis actitudes.

A mis hermanos quienes son los cómplices mis sueños; a Daniel, mi persona favorita y a Francisco por su invaluable conocimiento.

A mi abuelita y padrinos Dorita y Carlos, quienes siempre han estado ahí.

A mis ángeles Juan, Xime, Angie y Anita por haberme enrolado en el barco de la vida.

A mis amigos Gonzalo, Pato, Jhoa, Cris, Javi, Jimmy, Giss, Erick, Stalyn y Sofía, quienes han sido soporte a lo largo de mi estancia en la universidad.

A todos los docentes que cambiaron mi forma de ver las cosas. A mi director de Trabajo de Titulación Ing. David Mejía MSc. por la motivación, la paciencia y las oportunidades que me brindó para culminar este último paso en mi carrera de pregrado.

DEDICATORIA

A mi familia y amigos, a quienes no defraudaré en la contienda de cambiar al mundo.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE CÓDIGOS

RESUMEN

Este Trabajo de Titulación presenta un prototipo de aplicación distribuida para la gestión de contenidos en Alfresco y cuenta con un módulo gestor de metadatos, un módulo para sincronización, un módulo para aplicación de metadatos y un módulo de búsqueda basado en metadatos.

El módulo gestor de metadatos maneja modelos de metadatos personalizados para contenidos almacenados en Alfresco. El módulo de búsqueda permite buscar contenidos almacenados en Alfresco, según metadatos personalizados. El visualizador de archivos y carpetas permite la aplicación de metadatos sobre los archivos o carpetas. El módulo de sincronización permite replicar la estructura de árbol de archivos y carpetas de Alfresco dentro de un directorio del prototipo y mantenerlo sincronizado con su símil remoto.

Para el desarrollo se utilizó el ambiente de Windows Forms, el lenguaje de programación C# y a través de las interfaces RESTful de Alfresco se interactua con objetos de Alfresco como archivos, carpetas, metadatos personalizados.

En el primer capítulo se contextualiza los principales conceptos utilizados y se describen las herramientas utilizadas para el desarrollo del prototipo. El segundo capítulo presenta el proceso seguido para la implementación del prototipo; el levantamiento de requerimientos, diseño, implementación y validación, siguiendo las pautas de la metodología ágil de desarrollo de *software* Scrum. El tercer capítulo muestra los resultados de las pruebas realizadas, pruebas que muestran que los requerimientos se cumplen y que el usuario aprueba la aplicación; se realiza una retroalimentación a partir de los errores detectados. El cuarto capítulo muestra las conclusiones y recomendaciones de este proyecto.

Además, existen cinco anexos. En el primer anexo se muestra el modelo de encuesta que se utilizó para levantar información, en el segundo anexo se muestra el código de la aplicación, en el tercer anexo está detallado el diagrama UML de clases de la aplicación; el cuarto anexo muestra la encuesta de cierre y validación de la aplicación, y el quinto anexo presenta un manual de usuario.

PALABRAS CLAVE: Alfresco, API RESTful, metadatos.

ABSTRACT

The current project introduces a prototype of a distributed application for content management in Alfresco. This prototype has a metadata manager module, a synchronization module, a metadata addition module and a metadata-based search module.

The metadata manager module manages custom metadata models for content stored in Alfresco. The search module allows searching content stored in Alfresco, according to personalized metadata. The files and folders viewer allows the addition and edition of metadata on files or folders. The synchronization module allows to replicate the tree structure of Alfresco files and folders within some prototype directory and keep it synchronized with its remote simile directory.

For development the Windows Forms environment, the C# programming language and the Alfresco's RESTful interfaces were used. The interfaces let interact with Alfresco objects such as files, folders, and custom metadata.

In the first chapter, the main concepts used are contextualized and the tools used to develop the prototype are described. The second chapter presents the process followed for the implementation of the prototype; the requirements survey, design, development and validation, based on the guidelines of the agile methodology for software development Scrum. The third chapter shows the results of the tests performed, tests that show the requirements were met and the user approves the application; a feedback based on the detected errors were performed. The fourth chapter shows the conclusions and recommendations of this project.

Moreover, appendix section shows extend content which could not be written for its largeness. First appendix presents a survey that gather initial information about the application requirements; the second appendix shows the application code; the third appendix presents the UML class diagram of the application; the acceptation survey is showed in the appendix four and a user manual is introduced in the fifth appendix.

KEYWORDS: Alfresco, API RESTful, metadata.

1 INTRODUCCIÓN

El objetivo de este Trabajo de Titulación es desarrollar un prototipo de aplicación distribuida que permita la gestión de contenidos en Alfresco para la Escuela Politécnica Nacional. Este prototipo consta de un cliente que usa las interfaces de Alfresco y cuenta con un módulo gestor de metadatos, un módulo para sincronización, un módulo para agregación y edición de metadatos y un módulo de búsqueda basado en metadatos.

A través del módulo gestor de metadatos es posible crear, modificar y eliminar modelos de metadatos personalizados para archivos y carpetas almacenados en Alfresco. Este gestor se sustenta en una estructura de metagestión de contenidos llamada Modelo de Contenido.

También, este prototipo tiene un módulo de búsqueda basado en metadatos personalizados. Este módulo permite buscar contenido almacenado en Alfresco a través de una secuencia de filtros programables por el usuario dependientes del tipo de metadato.

Además, este prototipo posee un visualizador de archivos y carpetas almacenados en Alfresco que permite la aplicación de los metadatos, creados en el módulo gestor de metadatos, sobre los archivos o carpetas. Este módulo visualizador permite ver de manera estructurada archivos y carpetas, así como editar los metadatos genéricos y personalizados de los mismos.

Este prototipo también permite sincronizar contenido almacenado en la computadora de usuario con Alfresco. Esto permite replicar la estructura de árbol de archivos y carpetas de Alfresco dentro de un directorio del prototipo y verificar creaciones, modificaciones o eliminaciones del contenido del directorio para actualizar esas alteraciones en el servidor de Alfresco. De igual manera cualquier alteración del contenido en el servidor de Alfresco será actualizada en el directorio del prototipo.

A parte de ello, el prototipo permite la carga masiva de archivos y carpetas a través de la acción arrastrar y soltar.

Para la consecución de la aplicación se utiliza el ambiente de Windows Forms empleando el lenguaje de programación C# y a través de las interfaces RESTful de Alfresco es posible interactuar con objetos de Alfresco como archivos, carpetas, usuarios, metadatos y características personalizadas. También, mediante el uso de herramientas compatibles con el ambiente de programación es posible trabajar con contenido JSON para la serialización/deserialización del contenido proveniente de las interfaces. También se utilizan herramientas que permiten interactuar con metadatos de archivos y carpetas de Windows para lograr una sincronización entre la computadora de usuario y Alfresco.

1.1 Objetivos

El objetivo general de este Trabajo de Titulación es desarrollar un prototipo de aplicación distribuida para gestión de contenidos basado en Alfresco para la Escuela Politécnica Nacional.

Los objetivos específicos de este Trabajo de Titulación son:

- Analizar el funcionamiento del gestor de contenidos de Alfresco, así como sus esquemas de comunicación.
- Diseñar los módulos, clases y base de datos necesarios para la aplicación distribuida.
- Implementar la lógica y la presentación de la aplicación distribuida.
- Analizar los resultados obtenidos en las pruebas realizadas.

1.2 Alcance

En este Trabajo de Titulación se implementa una aplicación distribuida que actúa como cliente de Alfresco, la programación del cliente se la realiza en el ambiente de Windows Forms del lenguaje de programación C# y para la comunicación con el servidor Alfresco se utiliza un esquema de solicitudes y respuestas del tipo RESTful, tal como se muestra en la Figura 1.1. Además, se implementa una base de datos para guardar los parámetros de configuración de la aplicación.

La aplicación distribuida permite que un usuario realice cuatro acciones principales:

- Sincronización de contenido fuera de línea.
- Creación y adición de metadatos a los contenidos.
- Carga masiva de contenidos y visualización de contenidos y sus metadatos.
- Búsqueda basada en metadatos.

Las acciones anteriores implican la creación de los siguientes módulos principales: el primer módulo sirve para la sincronización del repositorio de usuario, este módulo se conecta al servidor y descarga todos los contenidos del usuario con sus respectivos metadatos; el segundo módulo sirve para crear metadatos, para ello, según Alfresco [1] primero es necesario crear un Modelo de Contenido, este Modelo de Contenido permite que Alfresco conozca, entre otras cosas, la diferencia entre un `int` y un tipo `date`, o entre

una carpeta y un archivo, este módulo permite crear paso a paso un Modelo de Contenido completo; el tercer módulo sirve para cargar contenidos en forma masiva y el cuarto módulo permite la búsqueda de contenidos haciendo uso de los metadatos.

Además, se crean otros sub-módulos secundarios como el módulo de acceso a la aplicación, el editor de metadatos de un archivo o carpeta de Alfresco y el navegador entre archivos y carpetas de Alfresco.

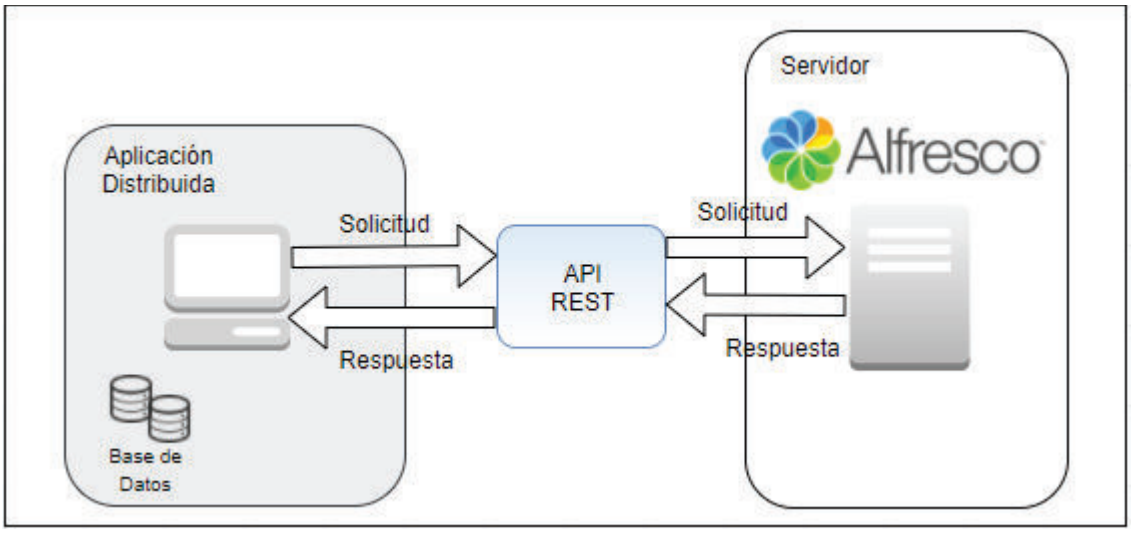


Figura 1.1 Esquema de integración entre una aplicación externa y Alfresco

1.3 Marco Teórico

1.3.1 Sistemas Distribuidos

Un sistema distribuido "es una colección de elementos computacionales independientes que aparecen al usuario como un solo sistema coherente" [2]. Un cliente navegando en una página web y accediendo a recursos remotos, es un ejemplo de un sistema distribuido. Los elementos computacionales que intervienen en un sistema distribuido pueden ser dispositivos *hardware* o procesos *software*, desde grandes computadoras de alto rendimiento hasta pequeños dispositivos de redes de sensores. De igual manera, la forma en que tales elementos están interconectados puede ser cualquiera, desde pequeñas redes LAN[1] hasta grandes redes WAN[2]. Tal interconexión permite que los elementos de un sistema distribuido interactúen entre sí, siguiendo una serie de estándares y protocolos comunes de tal manera que los elementos puedan entenderse unos con otros. Tal

[1] LAN (*Local Area Network*): Redes que operan dentro de un área limitada, ej: campus universitario.
[2] WAN (*Wide Area Network*): Redes que abarcan una extensa área geográfica.

interacción puede ser realizada de diversas formas, una de ellas es la utilizada en la web[3] mediante mensajes de solicitud y mensajes de respuesta entre dos computadoras.

Los usos que se puede dar a un sistema distribuido son muchos y dependen de la necesidad del usuario. Sin embargo, una de las necesidades más requeridas es la compartición de recursos [3]. Compartición de recursos físicos como impresoras o escáneres, o de recursos *software* como archivos y carpetas, son dos ejemplos claros de un sistema distribuido. Esta necesidad ha llevado a crear sistemas distribuidos que ofrezcan servicios de alojamiento y compartición de recursos, y permitan acceso a los mismos. Generalizando esta idea, los elementos computacionales que ofrecen servicios se les conoce como servidores y los elementos computacionales que acceden a tales servicios, clientes.

1.3.2 Modelo cliente-servidor

El modelo cliente-servidor en esencia constituye un modelo de redes de computadoras donde existe un elemento computacional que ofrece un servicio y otro o muchos elementos que consumen dicho servicio [3]. Este modelo es ampliamente conocido y constituye también un sistema distribuido.

La esencia de la computación cliente-servidor está basada en dos procesos, uno corriendo en la máquina cliente y otro corriendo en la máquina servidor, cuya comunicación se realiza mediante mensajes llamados solicitudes y respuestas. El elemento computacional encargado de servir generalmente es más robusto y debe soportar todas las posibles solicitudes de los diferentes clientes para luego procesarlas y responderlas. Mientras que un cliente generalmente resulta ser más ligero y generará solicitud y recibirá una respuesta. Sin embargo, un servidor puede convertirse en cliente si este servidor genera solicitudes hacia otro servidor. Por ello se dice que los términos cliente y servidor son aplicados como roles basados en una sola solicitud-respuesta [4]. La Figura 1.2 muestra un esquema de dispositivos que se comunican a través del modelo cliente-servidor.

La interacción de un sistema cliente-servidor requiere primero de una acción por parte del usuario (cliente), por ello es necesario que un sistema cliente-servidor presente la información para la creación y recepción de mensajes y genere la lógica por detrás para procesar solicitudes y producir respuestas. El procesamiento de solicitudes y generación de respuestas simplemente puede servir como un procedimiento, método o subrutina para el cliente o en muchos de los casos estas acciones necesitan ir acompañadas de cierta

[3] Web: Sistema de recursos conectados a través de Internet.

persistencia de la información. En ambos casos, la información que un cliente envía a un servidor puede contener lo que sea (más aun en sistemas abiertos) por lo que un sistema cliente-servidor debe contener (a parte de la presentación y procesamiento/persistencia) un mecanismo de gestión de los datos que pasan hacia y desde el cliente [4].

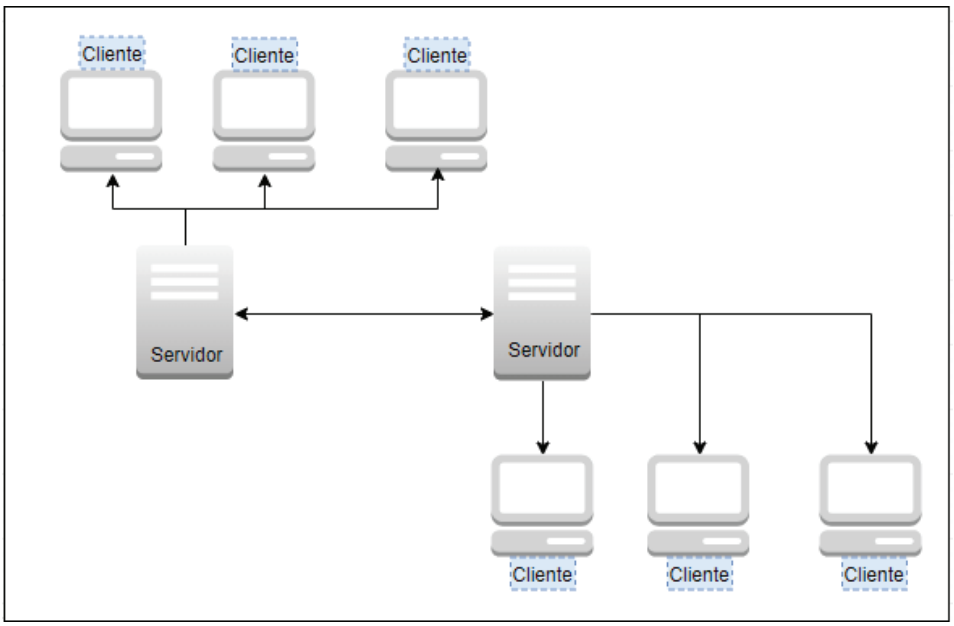


Figura 1.2 Ejemplo de disposición de dispositivos en el modelo cliente-servidor

1.3.3 HTTP

La interacción entre cliente-servidor en la web se realiza mediante un esquema de solicitudes y respuestas. Tales solicitudes y respuestas están estandarizadas de tal manera que exista comprensión entre cualquier cliente y cualquier servidor, y pertenecen a un esquema confiable (TCP[4]). HTTP (*Hypertext Transfer Protocol*) constituye el esquema de solicitudes y respuestas que permite lograr tal entendimiento.

HTTP implica que la primera interacción sea una solicitud dirigida a un computador que se encargará de procesar tal solicitud y enviar una respuesta. Para poder generar esa solicitud inicial, una de las opciones es acceder a una página web la cual es diferenciada por una dirección web. Esta dirección es llamada URL e identifica a un recurso y a un *host*[5]. Un servidor web puede hospedar muchas URL que en definitiva apuntarán a un determinado recurso en particular. Además, la URL identifica la dirección a la cual se realizará una solicitud HTTP.

[4] TCP (*Transport Control Protocol*): Es un protocolo de capa de Transporte según el modelo OSI. Permite comunicaciones confiables mediante un esquema mensaje-confirmación.
[5] *Host*: Nombre que se le asigna a un computador que forma parte de una red de computadoras.

Cuando la solicitud HTTP ha sido realizada, esta viajará a través de Internet para llegar hasta el servidor web. Allí será procesada y consecuentemente se enviará una respuesta HTTP que contendrá el contenido del recurso solicitado. Dicho recurso generalmente está escrito en un lenguaje estandarizado (HTML[6]) y deberá ser renderizado por el navegador para su presentación al usuario.

Desde un punto de vista práctico, la interacción cliente-servidor es representada en la Figura 1.3. En esta figura se observa que el encargado de generar la solicitud es el cliente quien necesita conocer la dirección web, está solicitud viaja a través de la red y llega al respectivo servidor para ser procesada y de tal procesamiento generar una respuesta, esta respuesta contiene el recurso solicitado y viaja a través de la red hacia el cliente, donde el cliente podrá observar el recurso.

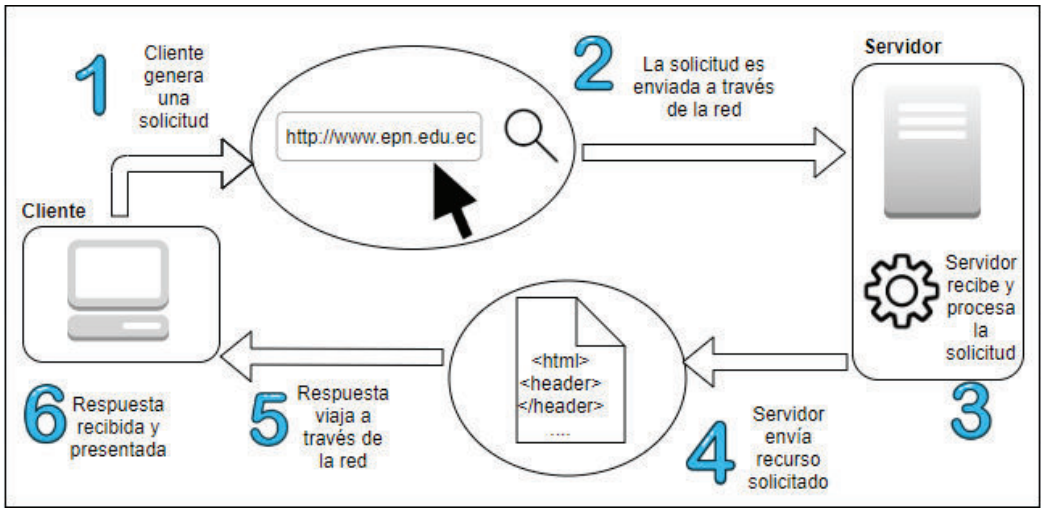


Figura 1.3 Interacción práctica entre cliente y servidor

URL (*Uniform Resorce Locator*): Es una dirección que identifica a un recurso y a un *host.* URL ha sido implementado como mecanismo principal de direccionamiento de recursos en la web. El esquema de una URL está conformado por cinco partes que permiten organizar el acceso a los recursos. En la Figura 1.4 se muestra la estructura de una URL.

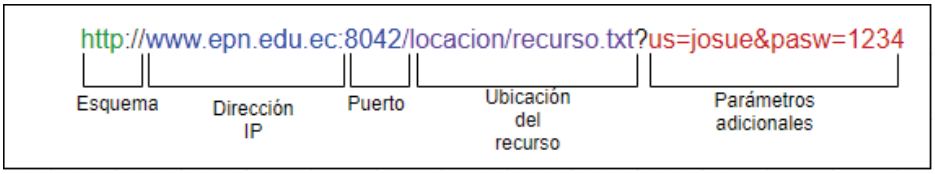


Figura 1.4 Esquema de una URL

[6] HTML (*HyperText Markup Language*): Es el lenguaje de marcado estándar para crear páginas web. Brinda una estructura y una presentación a la página web.

Esta estructura permite diferenciar la dirección y puerto del servidor al cual el cliente se conectará y la ubicación del recurso al cual se está accediendo. Es posible entonces alojar muchos recursos en un mismo servidor y diferenciarlos solo por la ubicación del recurso, tal como se muestra en la Figura 1.5. Esta ubicación termina identificando a un recurso, así como el nombre del recurso o la URL completa también identifican al recurso, se dice en general, que la cadena de caracteres que identifica a un recurso se denomina URI[7].

Siguiendo con el análisis de la Figura 1.4, la dirección IP[8] permite diferenciar al *host* en el cual se aloja el recurso y también se logra diferenciar el servicio que está ofreciendo el servidor mediante el puerto y el esquema. A parte de ello, es posible agregar parámetros adicionales que ejecutan una acción sobre el recurso, en el caso de la cadena de consulta de la Figura 1.4 `us=josue&pasw=1234` permitirá una autenticación básica del usuario `josue` con contraseña `1234`, embebida en la misma URL.

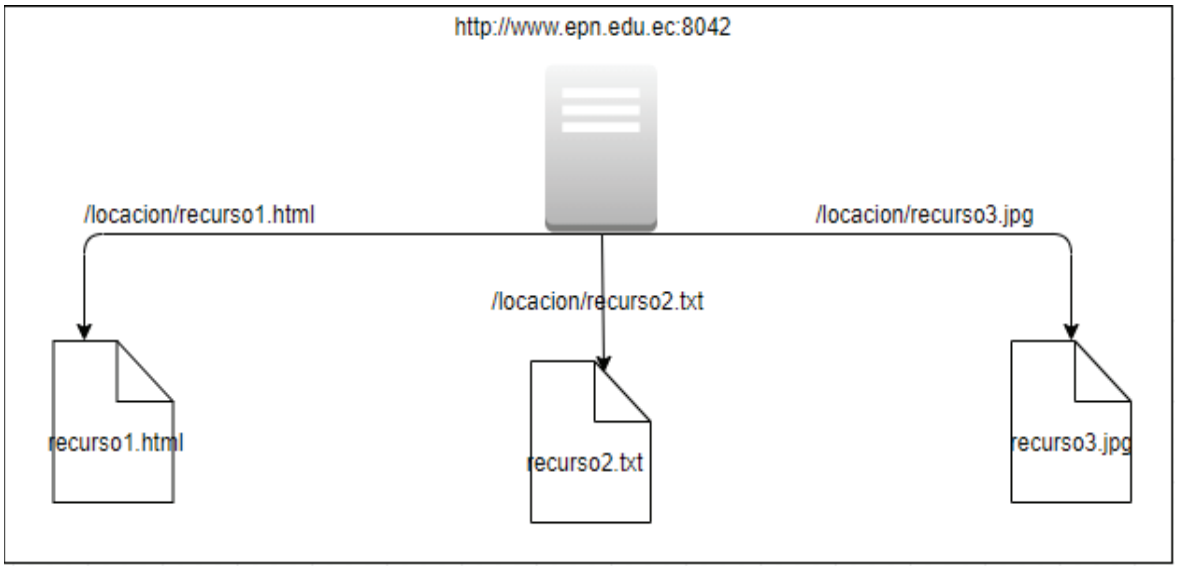


Figura 1.5 Disposición de recursos en un servidor

Solicitudes y respuestas HTTP: En la Figura 1.6 se muestra la estructura usada en las solicitudes y respuestas HTTP. La solicitud HTTP cuenta con un método que permite gestionar la solicitud misma, una URI que permite identificar a un recurso, la versión de HTTP que se está utilizando (por defecto HTTP 1.1) y las cabeceras que pueden contener información del cliente como autorizaciones de acceso a un recurso o tipo de recursos que se solicita. El cuerpo del mensaje de una solicitud contiene información adicional cuando

[7] URI (*Uniform Resource Identificator*): Cadena de caracteres que identifica a un recurso sin ambigüedad.
[8] Dirección IP: Dirección numérica asignada a cada dispositivo conectado a una red de computadoras la cual utiliza el protocolo de red IP.

la solicitud es de creación o modificación de un recurso. La respuesta HTTP en cambio, solo contiene la versión HTTP (por defecto HTTP 1.1), el código de estado y la razón los cuales proveen un informe del trato que se le dio a la solicitud subyacente, las cabeceras que usualmente mensajes del servidor al cliente y el cuerpo del mensaje en el cual viaja, generalmente, el recurso solicitado. El cuerpo del mensaje de una respuesta puede contener el recurso creado o modificado si se trata de la respuesta a una solicitud de creación o modificación de un recurso.

Figura 1.6 Estructura de una solicitud HTTP y de una respuesta HTTP

Métodos y códigos de estado HTTP: Las solicitudes HTTP son manejadas mediante métodos mientras que las respuestas son gestionadas mediante códigos de estado. HTTP cuenta con diversos métodos que permiten realizar solicitudes. Los métodos más utilizados son: `GET`, `POST`, `PUT` y `DELETE`. La Tabla 1.1 muestra los principales métodos HTTP, mientras que la Tabla 1.2 muestra el significado de algunos códigos HTTP [5], [6].

Cabe recalcar que los métodos y códigos de HTTP brindan una estructura estandarizada para lograr una comunicación entendible entre cliente y servidor.

Tabla 1.1 Uso de los principales métodos HTTP

Método	Uso
`GET`	Un cliente envía un `GET` para solicitar la representación de un recurso
`POST`	Un cliente envía un `POST` para crear un recurso
`PUT`	Un cliente envía un `PUT` para modificar el estado de un recurso
`DELETE`	Un cliente envía un `DELETE` cuando quiere eliminar un recurso
`OPTIONS`	Un cliente envía un `OPTIONS` cuando desea conocer a qué métodos HTTP responde un recurso

Tabla 1.2 Algunos códigos de estado de HTTP

Familia de códigos	Descripción	Códigos principales
2xx	Indican que la solicitud del cliente fue exitosamente recibida, entendida y aceptada.	200 *OK*
		201 *Created*
		204 *No Content*
3xx	Indican redirecciones o informaciones de caché	300 *Multiple Choices*
		301 *Moved Permanently*
		302 *Found*
		304 *Not Modified*
		307 *Temporary Redirect*
4xx	Indican errores en el lado del cliente	400 *Bad Request*
		401 *Unauthorized*
		403 *Forbidden*
		404 *Not Found*
5xx	Indican errores en el lado del servidor	500 *Internal Server Error*
		501 *Not Implemented*
		503 *Service Unavailable*
		550 *Permission denied*

Cabeceras*:* En la estructura de solicitudes y respuestas HTTP (ver Figura 1.6) se puede apreciar que tanto en las solicitudes como en las respuestas HTTP se encuentran presentes los campos cabeceras. Estos campos, a pesar de llevar el mismo nombre, no son iguales, es decir existen cabeceras de solicitud y cabeceras de repuesta. Las cabeceras de solicitud permiten al cliente enviar información adicional acerca de la solicitud y del cliente en sí al servidor. Mientras que las cabeceras de respuesta permiten al servidor enviar información que no ha podido tener lugar en el código de estado. También existen cabeceras propietarias las cuales permiten implementar una particular comunicación entre cliente y servidor. En la Tabla 1.3 se resumen algunas de las principales cabeceras de solicitud y respuesta [5], [7].

Tabla 1.3 Algunas cabeceras de solicitud y respuesta de HTTP

Cabecera	Definición
`Host`	Especifica el *host* y el puerto del recurso que se está solicitando.
`User-Agent`	Contiene información acerca de la aplicación, sistema operativo, proveedor de *software* del usuario que realiza la solicitud.
`Authorization`	Contiene las credenciales para autenticar a un usuario.
`Accept`	Es usado para especificar los tipos de datos que serán aceptados cuando responda el servidor.
`Last Modified`	Es usado para indicar la última modificación del recurso
`Content-type`	Indica el tipo del contenido del recurso enviado
`Content-lenght`	Indica el tamaño del contenido del cuerpo del recurso enviado

Estado y acceso a un recurso: Al momento de acceder a un recurso web mediante una solicitud `GET` este será el mismo para cualquier cliente que lo acceda. Una solicitud `GET` puede ser generada automáticamente desde un navegador web. En la Figura 1.7 se muestra una solicitud `GET` generada desde un navegador web y capturada mediante Wireshark[9], esta figura muestra la estructura de una solicitud `GET`, el método (`GET`), la URI (`/activity/5ae7eb56972f7a19ccb7ee59`), la versión HTTP (HTTP 1.1), todos separados por un espacio tal y como especifica el protocolo HTTP.

Las cabeceras que envía la solicitud `GET` de la Figura 1.7 están separadas por un salto de línea (`/r/n`) como especifica el protocolo y son las siguientes: `Host` que identifica al *host* (`localhost`) y al puerto de conexión (`3000`); `Connection`: `keep-alive` que indica que la conexión se mantendrá abierta para posteriores solicitudes, `Upgrade-Insecure-Request:1` que indica que el cliente soporta y prefiere una respuesta cifrada y autenticada; `User-Agent` que indica que el cliente desde el cual se generó la solicitud fue el navegador web Mozilla Firefox y el sistema operativo fue Windows NT 10 x64; `Accept` que indica que se aceptan respuestas del tipo `html`, `xml`, etc.; `Accept-Enconding` que indica que para la compresión se aceptan `gzip`[10] y `deflate`[11], y `Accept-Lenguage` que indica que el cliente acepta lenguaje español en un factor de calidad del 90% y lenguaje inglés en un factor de calidad del 80%. Al ser una solicitud `GET` esta no contiene cuerpo.

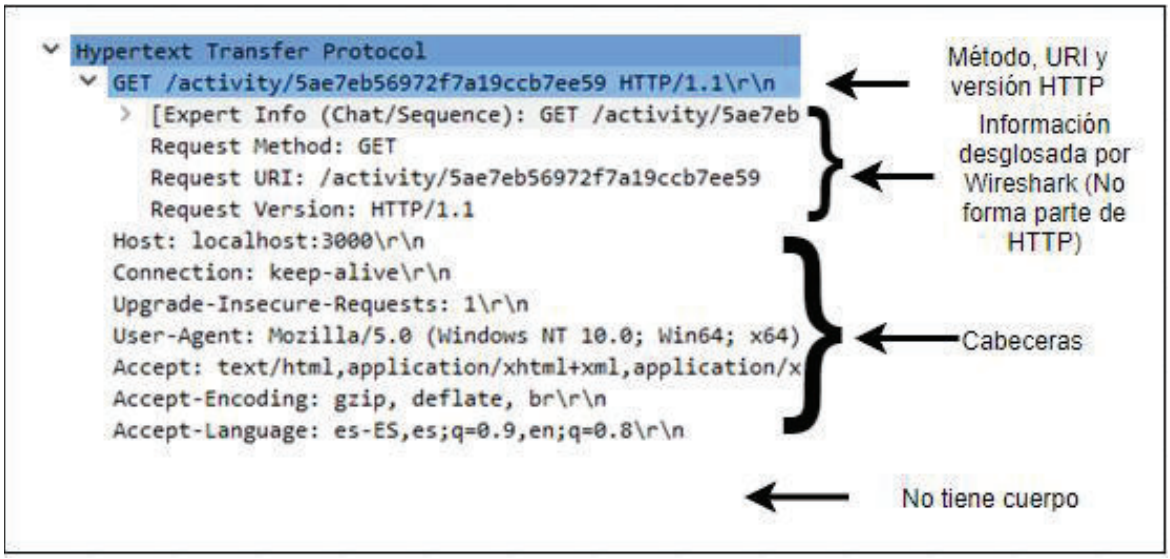


Figura 1.7 Solicitud `GET` capturada desde Wireshark

[9] Wireshark: Programa utilizado para capturar y analizar tráfico de una red.
[10] `Gzip`: Es un algoritmo de compresión de datos sin pérdida para un solo archivo. El archivo resultante tiene la extensión `.gz`.
[11] `Deflate`: Es un algoritmo de compresión de datos sin pérdida utilizado en la web.

La respuesta a la solicitud de la Figura 1.7 se detalla en la Figura 1.8 y se puede observar que se cuenta con la versión (HTTP 1.1) y el código de estado (`OK 200`, el cual significa que la respuesta ha sido aceptada por el servidor). Además en las cabeceras, están: `X-Powered-By` (cabecera no estandarizada) que indica que el servidor tiene la estructura `Express`[12] de Node.js[13]; `Content-Type` que indica que la respuesta es del tipo `application/json`; `Content-Lenght` que indica que el recurso solicitado tiene un tamaño de 244 `bytes`; `Etag` que indica un *token* e identifica al recurso enviado, como caché; `Date` que indica la fecha en la que se generó la respuesta; y `Connection keep-alive` que indica que la conexión se mantiene abierta para posteriores solicitudes. El recurso solicitado viaja en el cuerpo de la respuesta y se puede observar que es un objeto JSON (`application/json`) con los atributos `name`, `description`, `url_how_to_do`, `level`, `muscle_group`, `implements`, `_id` y `_v`.

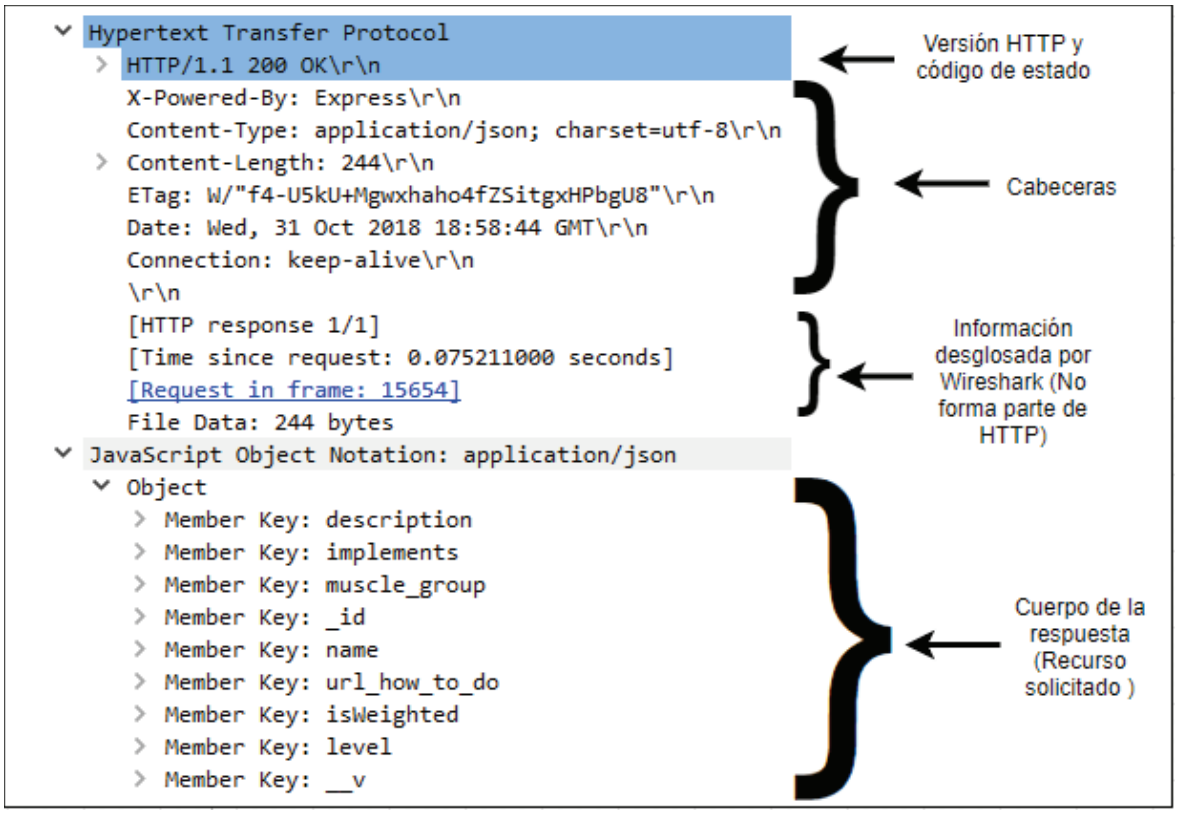


Figura 1.8 Respuesta HTTP a un `GET`

En contraparte, hacer un `POST`, `PUT` o `DELETE` puede hacer que los recursos cambien. Analizando el método `POST`, en la Figura 1.9 se muestra una solicitud `POST` generada

[12] Express: Infraestructura de aplicaciones web de Node.js.
[13] Node.js: Entorno de ejecución para Javascript.

desde Postman[14], apuntada hacia la URI `/activity` con la versión HTTP 1.1. Esta solicitud contiene cabeceras entre las que destacan: `Content-Type` que indica que la solicitud es del tipo `application/json`; `cache-control` que indica que el cliente puede almacenar en caché la respuesta; `Postman-Token` (cabecera personalizada) que se usa para evitar un *bug*[15] del navegador web Google Chrome; `User-Agent` que indica que el agente que envía la solicitud es PostmanRuntime/7.3.0 (Nombre genérico de Postman); `Accept` que indica que se acepta cualquier tipo de datos de respuesta; `Host` que indica que el *host* es `localhost` y el puerto de conexión es el `3000`; `accept-encoding` que indica que para la compresión se acepta `gzip` y `deflate`; `content-lenght` que indica el tamaño del contenido del cuerpo de la solicitud, en este caso 209 `bytes` y `Connection keep-alive` que indica que la conexión se mantiene abierta para posteriores solicitudes.

En el cuerpo de esta solicitud se puede notar que el contenido es del tipo `application/json` y contiene un objeto con los atributos `name`, `description`, `url_how_to_do`, `level`, `muscle_group`, `implements`.

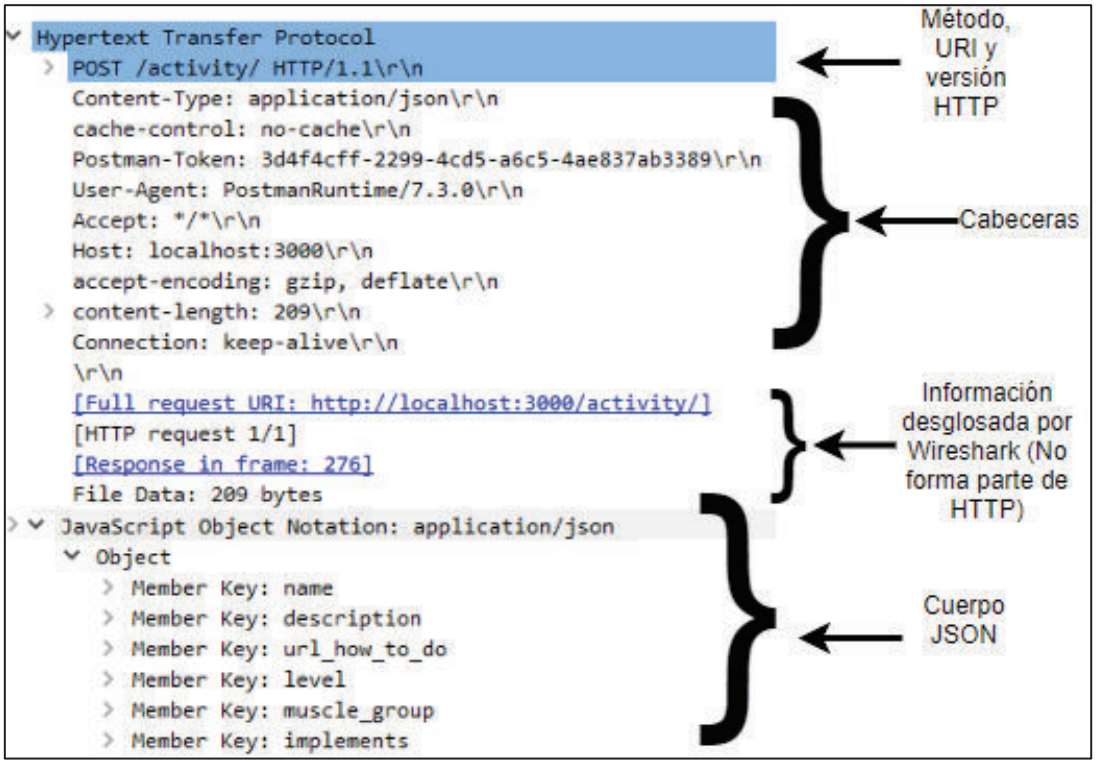


Figura 1.9 Solicitud `POST` generada desde una aplicación web

[14] Postman: Programa de escritorio, desarrollado para construir y probar una API.
[15] *Bug*: Error sin corregir de un programa o sistema informático.

Por otra parte, la respuesta a la solicitud de la Figura 1.9 es similar a la respuesta de un `GET` (Figura 1.8). Sin embargo, cabe destacar que el cuerpo de la respuesta para un `POST` puede contener el recurso creado o simplemente ninguna información.

Finalmente, cuando se genera una solicitud `GET`, `POST`, `PUT` o `DELETE` desde un navegador, el proceso es trasparente, es decir, solo basta con especificar la URL o hacer clic en algún botón. Sin embargo, si se requiere crear una solicitud más personalizada, existen programas como Postman, cURL[16] o Advanced REST Client[17], que logran permitir controlar los diferentes campos y parámetros de una solicitud.

1.3.4 Servicios web

Un Servicio web provee una interfaz de servicio para que un cliente interactúe con un servidor de una manera más específica de la que lo hacen los navegadores web.

API (*Application Programming Interface*): Una API provee un mecanismo a los desarrolladores para acceder a datos y servicios, y poder construir aplicaciones. Una API es en esencia un contrato entre un proveedor y un consumidor, este contrato permite al consumidor utilizar un servicio o acceder a un recurso a través de una interfaz bien documentada, consistente y predecible [8].

La forma de acceder a una API puede ser muy variada. Por ejemplo, Java[18] ofrece la API JDBC[19] que provee un acceso universal a datos SQL[20] desde Java [9]. Esta API comprende dos paquetes Java que funcionan como librerías de forma local. En contra parte, una API web es caracterizada por que su acceso no es local, sino es realizado a través de la web, por ejemplo, a través de HTTP.

Usualmente, una empresa grande (proveedor) extiende una API pública o privada para que una empresa más pequeña (consumidor) añada los servicios del proveedor a sus propios servicios. Un ejemplo es la API que ha extendido la red social Twitter. Esta API permite publicar y analizar *tweets*, optimizar anuncios y crear mejores experiencias de usuario [10]. Una empresa que consume esta API es Next Analytics[21]. Esta empresa utiliza la API de Twitter para obtener información de seguidores, seguidos, estados y *hashtags*[22]. Estos

[16] cURL: Programa (Windows) o comando (Linux) usado para transferir datos.
[17] Advanced REST client: Extensión de Chrome para probar y crear solicitudes HTTP personalizadas.
[18] Java: Lenguaje de programación orientado a objetos.
[19] JDBC (*Java Data Base Connectivity*): Es una interfaz de programación de aplicaciones que permite que el lenguaje Java acceda a sistemas de bases de datos de una manera universal.
[20] SQL: Lenguaje universal de acceso para bases de datos relacionales.
[21] Next Analytics: Empresa enfocada en la analítica de datos.
[22] *Hashtag*: Cadena de caracteres sin espacios precedidos por el caracter almohadilla (#).

datos son procesados y luego formateados con el fin de generar datos valiosos que permitan mejorar estrategias y técnicas de ventas. En la Figura 1.10 se muestra la estructura típica del uso de una API [10], donde se observa que los desarrolladores usan la API para crear nuevas aplicaciones con servicios combinados (del proveedor y nuevos).

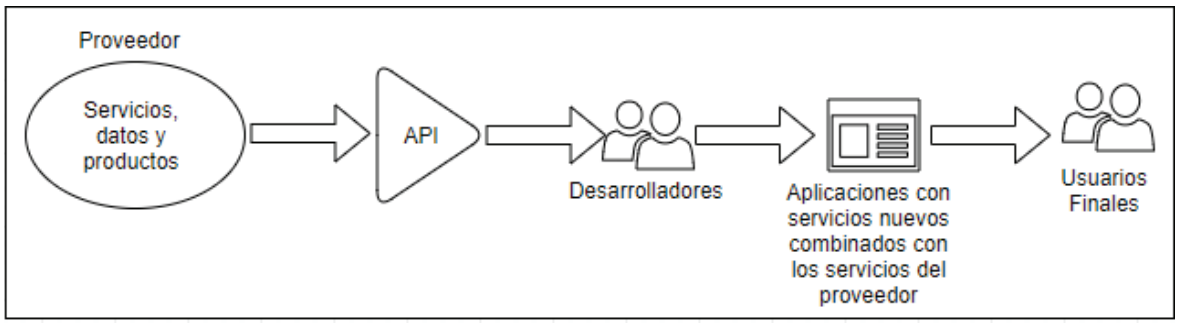


Figura 1.10 Estructura genérica del uso de una API

API RESTFul: Para definir este término es necesario comprender el significado de REST. REST es un estilo arquitectónico basado en una serie de principios que definen la disposición de los recursos web o definen la transferencia de recursos de un servidor a un cliente [4]. El término RESTful se refiere a un servicio que implementa la arquitectura REST [11]. Una analogía para comprender de mejor manera estos términos es la de la arquitectura Orientada a Objetos (análoga a REST) y una aplicación que implementa tal arquitectura (análoga a RESTful) [12]. A breves rasgos un recurso en una API RESTful se identifica mediante una URI y se comunica a través de los métodos `GET`, `POST`, `PUT` y `DELETE` de HTTP.

El uso de una API RESTful se caracteriza por su simplicidad, ya que basta con comprender la documentación ofertada por el proveedor de la API, dominar el uso de HTTP y obviamente conocer las reglas de negocio del cual se requiere extender la API. Sin embargo, la complejidad se basa en decodificar los recursos accedidos y codificar de manera correcta los recursos a crearse o modificarse. Esta codificación/decodificación para RESTful es tratada por formatos para el intercambio de datos como XML[23] o JSON. Ambos formatos ofrecen una estructura para los datos de tal manera que puedan ser entendidos por computadores y humanos.

JSON (Javascript[24] *Object Notation*): Es un formato de intercambio de datos liviano basado en el lenguaje de programación Javascript. A pesar de su nacimiento a partir de

[23] XML (*eXtensible Markup Language):* Lenguaje utilizado para transferencia y persistencia de datos.
[24] Javascript: Lenguaje de programación mayormente utilizado para programación web.

Javascript esta notación es totalmente independiente de cualquier lenguaje de programación y puede ser usado en cualquiera de ellos [13].

JSON está construido sobre dos estructuras, la primera, los objetos los cuales son una colección de pares tipo clave/valor; mientras que la segunda estructura es una lista de valores ordenados o mejor conocido como *array*. Para describir objetos se utilizan llaves y comillas dobles: `{"clave": "valor"}`. Mientras que para describir listas o *arrays* se utiliza corchetes: `[1, 2, 3]`. Estos dos conceptos pueden ser mezclados y en realidad es posible tener incluso objetos conformados por *arrays* o *arrays* conformados por valores y objetos. En la Figura 1.11 se muestra un ejemplo de estructura JSON, los atributos del objeto son `name`, `description`, `url_how_to_do`, `level`, `muscle_group` e `implements`. El atributo `implements` es del tipo *array*.

```
{
    "name":"Prueba Tesis",
    "description":"No description",
    "url_how_to_do":"https://www.youtube.com/watch?v=TU8QYVW0gDU",
    "level":"Advanced",
    "muscle_group":"Shoulders",
    "implements":["Body weight"]
}
```

Figura 1.11 Ejemplo de un objeto JSON

Como se puede observar en la Figura 1.11, los objetos JSON están basados en los `string` de doble comilla, es decir tanto las claves como sus valores deben estar entre comillas dobles y separados por dos puntos; para separar pares se utilizan comas.

1.3.5 Alfresco Community

Alfresco Community es un sistema de código abierto ECM (*Enterprise Content Management*) que gestiona el contenido digital dentro de una empresa o institución y provee servicios y controles para gestionar estos contenidos [14]. Alfresco Community pertenece a uno de los grandes paquetes ofertados por Alfresco Inc., los otros dos constituyen la versión *premium* y la versión *cloud*.

La función principal de Alfresco Community es persistir contenido compartido, así como agregarle metadatos y crear asociaciones entre ese contenido. Sin embargo, Alfresco Community también ofrece gestión de procesos de negocio (BPM[25]) y soporta trabajo en equipo a través de mini repositorios llamados sitios. A parte de ello, una característica

[25] BPM: *Business Process Management*, permite cumplir procesos empresariales mediante la asignación y supervisión de tareas.

importante es que Alfresco provee interfaces de programación que soportan muchos lenguajes de programación y protocolos sobre los cuales un desarrollador puede crear aplicaciones clientes personalizadas, tal como lo muestra la Figura 1.12, en la cual se observa que aplicaciones personalizadas pueden ser conectadas al *core* de Alfresco a través de CMIS[26] o REST.

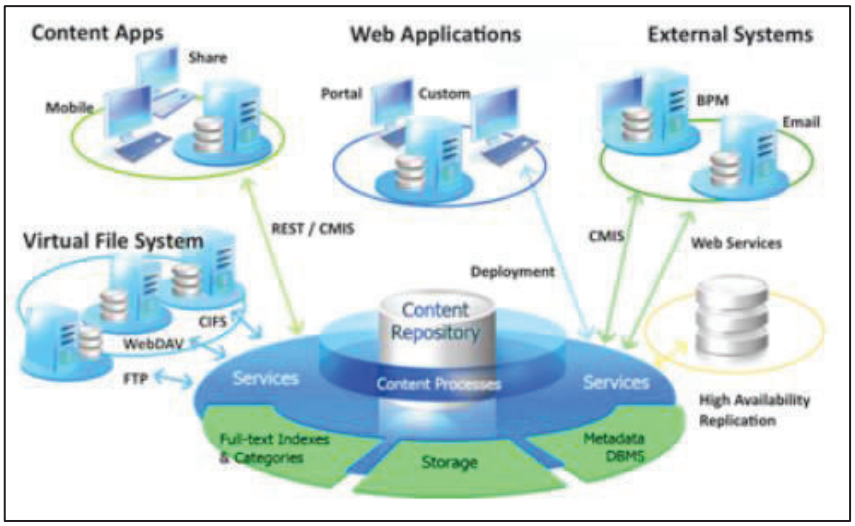


Figura 1.12 Integración de aplicaciones externas con Alfresco Community [14]

Alfresco Share: Alfresco Share es la interfaz web que provee Alfresco Community y que provee una comunicación entre el servidor de Alfresco y el usuario. Esta interfaz web permite al usuario ingresar al sistema, observar sus propios contenidos y contenidos compartidos, trabajar en equipo y gestionar tareas de procesos empresariales. Alfresco Share está disponible por defecto en la URI `localhost:8080/share/page`. La Figura 1.13 (a) muestra la página inicial de inicio de sesión de Alfresco Share y la Figura 1.13 (b) el *dashboard* de usuario.

Alfresco Share cuenta con seis pestañas principales, estas se muestran en la parte superior de la Figura 1.13 (b) y permiten: acceder al contenido en el propio repositorio (Mis ficheros), al contenido compartido (Ficheros compartidos), a los sitios a los que pertenece el usuario (Sitios), a las tareas asignadas al usuario (Tareas), a la búsqueda de personas (Personas) y al tablero principal o *dashboard* (Inicio). Un sitio es un área de proyecto donde se puede compartir y comentar contenido con otros miembros del sitio. Una tarea es una asignación que se le ha dado al usuario, puede ser, por ejemplo, la revisión de un documento o la aceptación de la invitación a un sitio.

[26] CMIS (*Content Management Interoperability Services*): Estándar para interoperación entre varios gestores de contenido a través de internet.

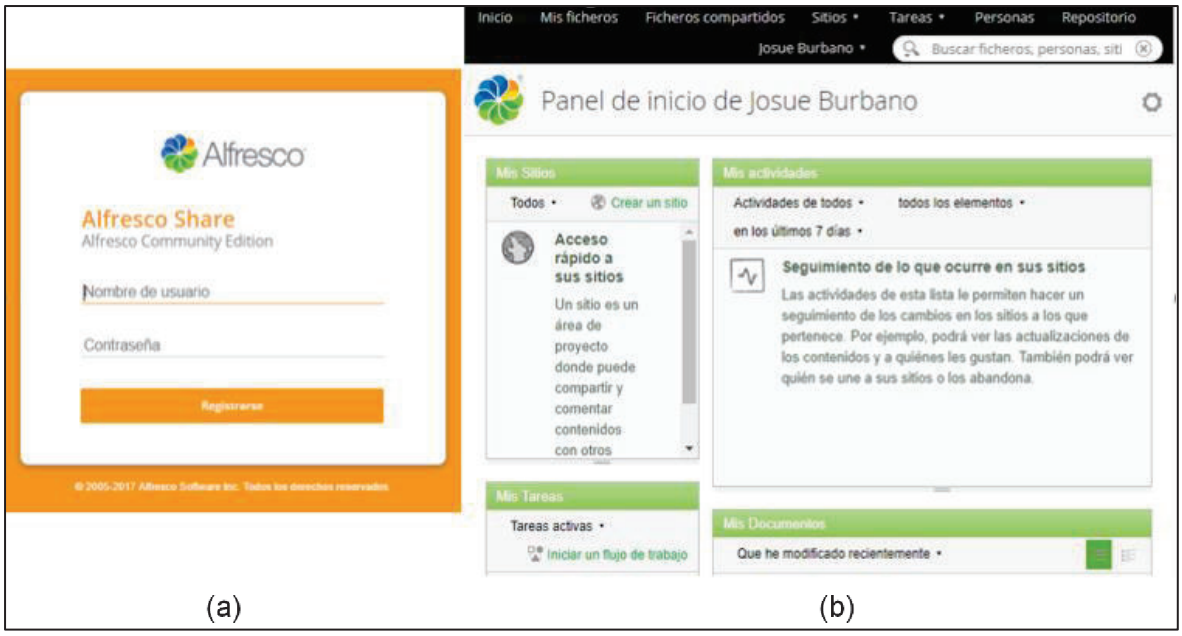


Figura 1.13 Inicio de sesión (a) y *dashboard* de usuario (b) de Alfresco Share

Repositorio: El repositorio de Alfresco es donde se encuentran tanto archivos propios del usuario como compartidos. La estructura de los contenidos está organizada en carpetas y archivos. Alfresco Share permite intuitivamente crear carpetas y cargar archivos, así como también permite crear contenido como texto plano, HTML, XML, documentos de Google Docs o desde una plantilla personalizada. Es posible también editar contenido existente en línea, a través de Google Docs, o fuera de línea, para ello Alfresco Share provee una herramienta de bloqueo de contenido cuando este se está modificando, una vez que se ha terminado la modificación basta con desbloquear el documento, versionar el mismo y si lo requiere volver a cargar el documento actualizado para que este vuelva a su estado de compartición normal.

Administrador: El administrador de Alfresco posee capacidades superiores a todos los usuarios además de ser el primer usuario de Alfresco. En Alfresco Share, este usuario posee acceso a la pestaña Herramientas de administración y en esta pestaña el administrador puede entre otras cosas crear filtros de búsqueda, administrar etiquetas[27], administrar sitios, crear usuarios y grupos de usuarios, y gestionar Modelos de Contenido.

1.3.6 Modelos de Contenido

Uno de los puntos fundamentales de un ECM son los metadatos [15], [16]. Alfresco provee un mecanismo de metadatos por defecto, que entre los principales se tiene, nombre, título

[27] Etiqueta: Metadato general asociado a un contenido.

y descripción. Sin embargo, la potenciación de Alfresco está en los metadatos personalizados y aunque Alfresco sí provee las herramientas para crear nuevos metadatos, esta manera de crear metadatos es centralizada y causa sobrecargas en el administrador, en especial para compartición de contenido en proyectos o departamentos, donde existe un encargado, y es él quien debería definir la estructura de los metadatos.

Desde una perspectiva más profunda la estructura de los metadatos está contenida en los Modelos de Contenido. Los Modelos de Contenido permiten, entre otras cosas, diferenciar entre una carpeta y un archivo o entre un dato que deba almacenarse como tipo `int` y un dato que requiera almacenarse como tipo `date` [17]. De hecho, un Modelo de Contenido es tan fundamental que es la gran brecha entre un gestor de contenidos y una carpeta compartida en red.

Técnicamente, un Modelo de Contenido constituye una estructura que define una estructura de metadatos compleja, es decir, un contenido puede tener la Propiedad (metadato personalizado) llamada “Fecha de Expiración” el cual tendrá asociado un valor de fecha para cada contenido, este valor puede estar definido como tipo `date` y puede estar restringido a las fechas 12/12/2000 y 12/12/2020. Esas definiciones y restricciones, así como otras acciones más, son descritas dentro del Modelo de Contenido.

Las Propiedades que se pueden generar a través de un Modelo de Contenido están estructuradas en dos grupos: Tipos y Aspectos. Un nodo (archivo o carpeta) puede ser de un solo Tipo, pero puede tener varios Aspectos. La Figura 1.14 muestra la jerarquía de los Modelos de Contenido.

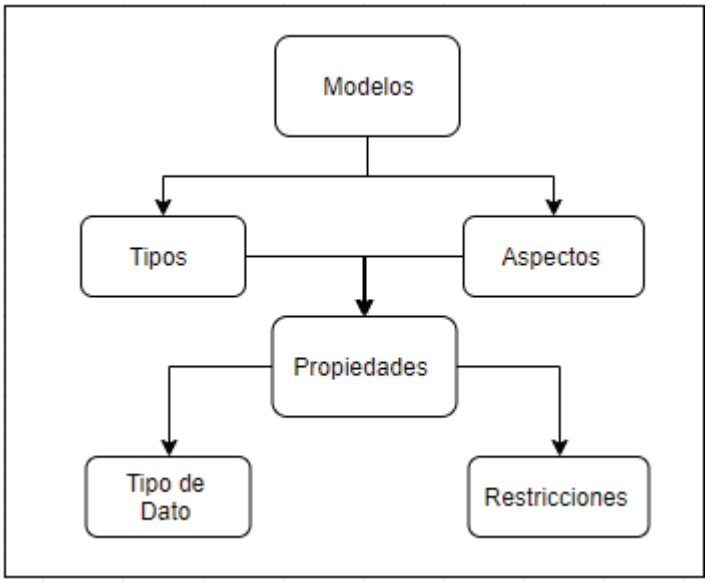


Figura 1.14 Estructura de meta-gestión mediante Modelos de Contenido de Alfresco [18]

Tipos: Los Tipos que ofrece Alfresco son análogos a las clases en el mundo orientado a objetos. En Alfresco existen tres Tipos principales, Contenido, Persona y Carpeta, y también es posible crear Tipos personalizados. Un Tipo personalizado tiene "atributos" llamados Propiedades. La Figura 1.15 muestra un ejemplo de Tipos y Propiedades personalizadas [17], donde el Tipo representa la clase y las propiedades los atributos. Un nodo solo puede ser de un solo Tipo.

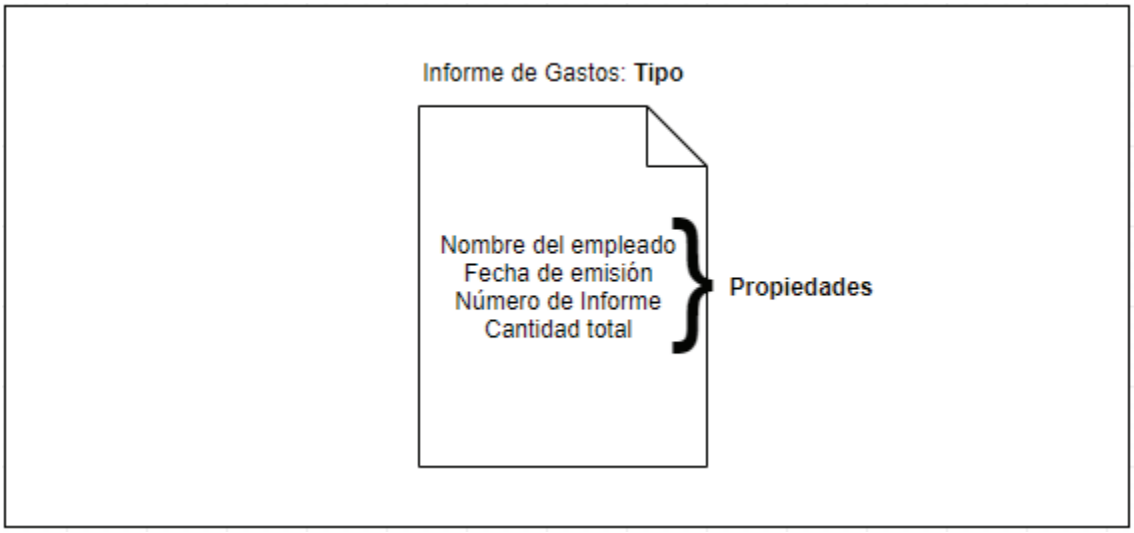


Figura 1.15 Ejemplo de Tipo y Propiedades personalizadas

Aspectos: Permiten compartir una Propiedad entre muchas clases, por ejemplo, "versionable" o "comentable". Los Aspectos no necesariamente son aplicados a archivos o carpetas del mismo Tipo, sino puede ser aplicados a archivos y carpetas de cualquier Tipo, es decir, un Aspecto constituye un conjunto de Propiedades de aspecto global.

Un ejemplo se muestra en la Figura 1.16, ahí se puede apreciar que el Aspecto `Aprobable` consta de dos Propiedades, `Aprobable` que es del tipo de dato `bool` y `Fecha de Aprobación` que es del tipo de dato `date`. Este Aspecto `Aprobable` ha sido aplicado a dos nodos de diferente Tipo, tales nodos heredan las Propiedades del Aspecto y tienen valores distintos para cada nodo. Un nodo (archivo o carpeta) puede tener muchos Aspectos.

Propiedades: La base de un Modelo de Contenido está sobre los Tipos, las Propiedades y los Aspectos. Sin embargo, las Propiedades constituyen la parte compleja de un Modelo de Contenido. Una Propiedad, aparte de contener el valor del metadato en sí (ya que Modelos, Tipos y Aspectos son abstractos), especifica el tipo de dato (`Integer`, `String`, etc.) y restricciones como lista de valores permitidos, multiplicidad, requerido, expresiones regulares, longitud y tamaño.

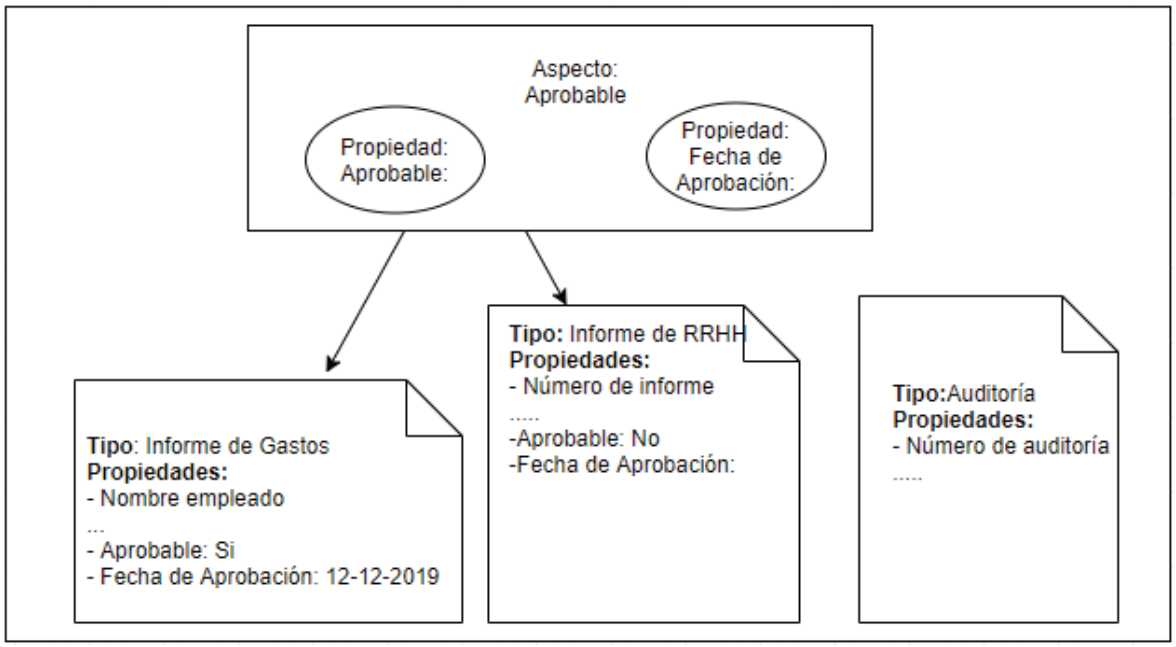


Figura 1.16 Ejemplo de Aspecto

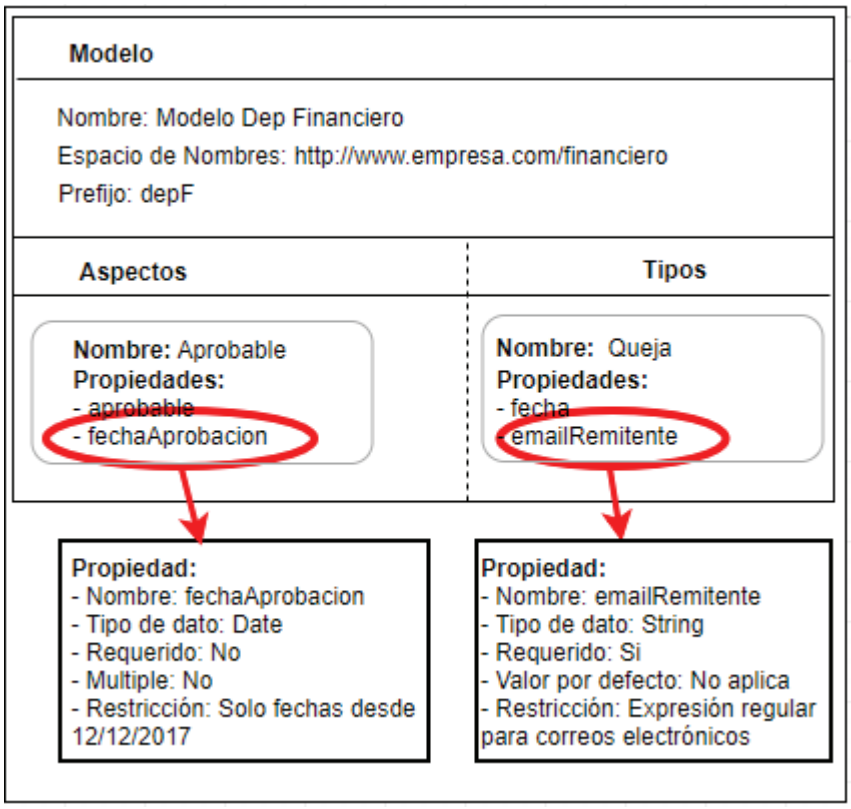


Figura 1.17 Ejemplo de Modelo, Aspecto y Propiedades

En la Figura 1.17 se muestra un ejemplo de toda una estructura de Modelo de Contenido haciendo énfasis en dos propiedades. La primera propiedad, `fechaAprobacion`, perteneciente al aspecto Aprobable y de tipo de dato `Date`, es opcional, es un solo valor y

solo es posible ingresar fechas mayores al 12/12/2017. La segunda propiedad, `emailRemitente`, es de tipo de dato `String`, es obligatorio y el valor de la propiedad debe encajar con la expresión regular especificada.

Por otra parte, es necesario nombrar que los Modelos de Contenido están cimentados en el lenguaje XML, por lo que un administrador podría crear o modificar ciertos archivos y crear un Modelo de Contenido.

Hay que tener en cuenta que el lenguaje XML maneja Espacios de Nombres para diferenciar contextos. Estos Espacios de Nombres identifican al Modelo de Contenido y están conformados por una URI y un prefijo. Aunque estas URI deberían ser usadas sólo para dar un nombre único a un Modelo de Contenido, generalmente se acostumbra a apuntarlas a algún sitio web donde se detalle el Modelo de Contenido [19]. Un prefijo está asociado a un espacio de nombres y es lo que precede al nombre de una etiqueta dentro de XML para diferenciar contextos.

Modelo de Contenido por defecto: Un nodo (archivo o carpeta) es un objeto de Alfresco (hablando en términos de Programación Orientada a Objetos). Este objeto almacena sus metadatos dentro de un *array* de nombre `properties`, y este *array* contiene metadatos para la descripción, el título, la versión y la fecha de modificación del nodo (ver Figura 1.18 (a)).

La Figura 1.18 (a) muestra algunos de los atributos de un nodo: `Name`, `NodeType`, `IsFile`, `IsFolder`, `AspectNames` y `Properties`. El valor del atributo `NodeType` es `cm:content` es decir este nodo es del Tipo por defecto; los valores del atributo `AspectNames` son `cm:titled`, `cm:versionable` y `cm:auditable` que son los aspectos por defecto; los valores del atributo `Properties` son `cm:description`, `cm:title`, `cm:version` y `cm:modified`, estos valores son derivados de los Aspectos y del Tipo del nodo, es decir, el nodo hereda las Propiedades del Tipo al que pertenece y de sus Aspectos.

La Figura 1.18 (b) muestra como está definido el Modelo de Contenido de ese nodo. El Modelo cuenta atributos como Nombre, Espacio de Nombres y Prefijo que lo distinguen de otros Modelos y tiene definido un Tipo (`cm:content`) y tres Aspectos (`cm:titled`, `cm:versionable`, `cm:auditable`). Los atributos de cada Aspecto están definidos con un Nombre y sus respectivas Propiedades. Estas definiciones sirven para saber qué propiedades contendrá un nodo cuando un Aspecto o un Tipo sea aplicado sobre tal nodo. Cabe destacar que las Propiedades definidas en la Figura 1.18 (b) solo constan como

nombres y tienen por detrás una estructura mucho más compleja, similar a lo que se muestra en la Figura 1.17. Todas las definiciones del Modelo de Contenido por defecto están contenidas en el archivo `contentModel.xml` perteneciente a Alfresco.

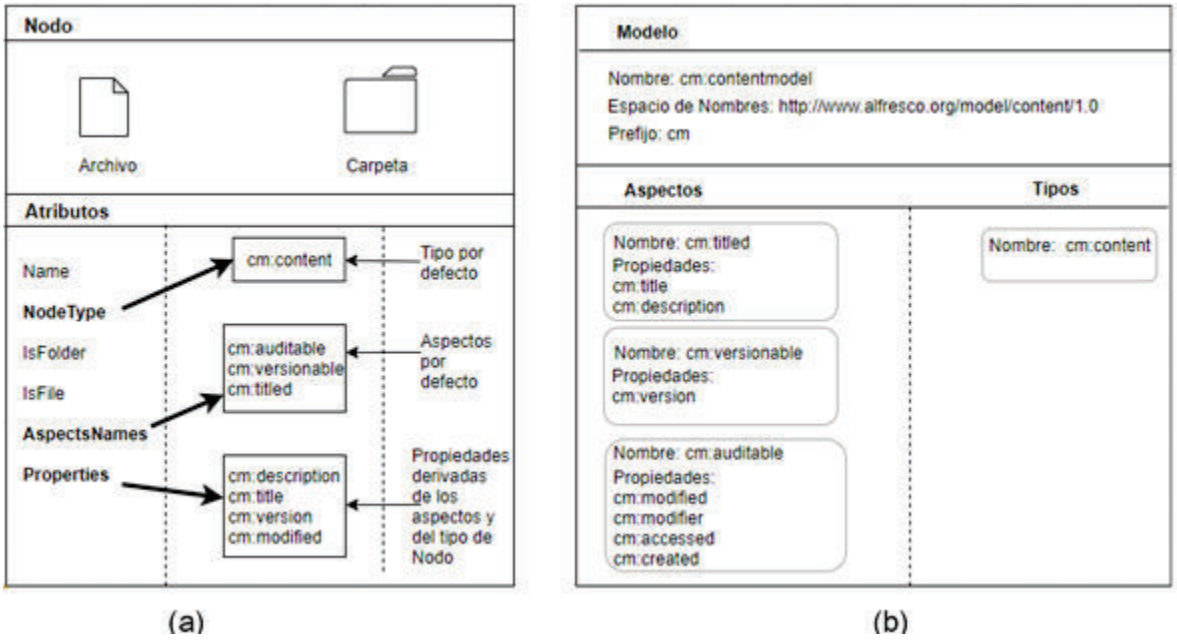


Figura 1.18 Modelo de Contenido por defecto

1.3.7 Alfresco: Integración de Software

Alfresco ofrece muchas opciones de desarrollo y personalización. Existen opciones que permiten trabajar sobre la lógica del servidor, otras que permiten trabajar como extensiones a la lógica del servidor y otras que permiten desarrollar clientes independientes del servidor. Dentro de las opciones que permiten manejar clientes independientes del servidor están la API RESTful y la CMIS API RESTful.

La integración de aplicaciones externas mediante la API RESTful de Alfresco resulta mucho más personalizada que la integración mediante CMIS API RESTful debido a que la primera pertenece explícitamente a Alfresco. Por otra parte, la CMIS API RESTful es más genérica debido a que es un estándar creado para la compatibilidad entre diferentes gestores de contenidos que provee una estructura para asociar diferentes metamodelos de contenido de diferentes proveedores.

CMIS API RESTful: El estándar CMIS define un modelo de dominio y provee enlaces RESTful que las aplicaciones pueden usar para gestionar sistemas de gestión de contenidos. Además, CMIS está apoyando por el proyecto Apache Chemistry[28] el cual

[28] Apache Chemistry: Grupo de proyectos con implementaciones de código abierto para CMIS.

ofrece librerías cliente, que encapsulan el acceso HTTP a través de objetos, para Java, Python[29], PHP[30], C#, Objective-C[31] y Javascript. Sin embargo, cada sistema ECM es el encargado de asociar su modelo de dominio al modelo de dominio provisto por CMIS. Si un sistema ECM posee una característica no basada en CMIS entonces no será posible asociar directamente tal característica a una del modelo de dominios de CMIS.

A nivel de cliente, CMIS ofrece varios servicios que permiten interactuar con un sistema ECM. Entre ellos están los servicios de repositorio que permiten descubrir repositorios disponibles y proporcionar información básica sobre ellos. También están los servicios de navegación y de objetos mediante los cuales se puede acceder a la jerarquía de contenidos, y realizar operaciones básicas como crear, leer, actualizar y eliminar objetos (contenidos) así como acceder a sus propiedades (metadatos). Otro de los servicios importantes que ofrece CMIS es el de búsqueda, cambios en los contenidos y versionamiento de los mismos. En contra parte, CMIS no soporta directamente operaciones de creación, modificación y eliminación de Modelos de Contenido debido a que los Modelos de Contenido son una característica particular de Alfresco.

Alfresco API RESTFul: Alfresco provee algunas API RESTful que permiten interactuar con los servicios y contenidos de Alfresco. Aunque oficialmente [20], Alfresco provee cinco API públicas existen otras API privadas [21] que Alfresco ofrece y que en conjunto logran permitir acceso suficiente para administrar contenido. En la Tabla 1.4, se resume el uso y los *end-points* asociados a las principales API.

Core API: Es la API más grande y proporciona acceso y manejo de personas, actividades, nodos y sitios. De los dieciocho grupos que la conforman los más notables son: *nodes, people* y *queries.* Cada acción viene acompañada de su uso, es decir, que parámetros utiliza y que contiene el cuerpo de cada solicitud y respuesta. En esta API se manejan objetos JSON. Muchas de las acciones que un nodo tiene, se resumen en el grupo *nodes* de la Core API, tal como se muestra en la Tabla 1.5. Para el manejo de personas, los *end-points* están organizados en el grupo *people*, tal y como se muestra en la Tabla 1.6. Mientras que para una búsqueda simple de nodos se utiliza el grupo *queries*, detallado en la Tabla 1.7.

El manual de cada acción (parámetros, cuerpo y respuestas) se encuentra especificado en [20].

[29] Python: Lenguaje de programación interpretado de alto nivel.
[30] PHP: Lenguaje de programación utilizado en su mayoría en la web.
[31] Objective-C: Lenguaje de programación orientado a objetos basado en C.

Tabla 1.4 API de Alfresco

API	URI	Uso
Core API	`/alfresco/api/-default-/public/alfresco/versions/1`	Proporciona acceso a las funciones principales de Alfresco.
Authentication API	`/alfresco/api/-default-/public/authentication/versions/1`	Proporciona acceso a las funciones de autenticación de Alfresco.
ContentCMN API	`/alfresco/api/-default-/private/alfresco/versions/1`	Proporciona acceso a las características de administración de contenido de Alfresco.
Content Search API	`/alfresco/api/-default-/public/search/versions/1`	Proporciona acceso a las funciones de búsqueda de Alfresco.

Tabla 1.5 Descripción del grupo *nodes* de la Core API

Método	URI	Acción
`DELETE`	`/nodes/{nodeId}`	Eliminar un nodo
`PUT`	`/nodes/{nodeId}`	Actualizar un nodo
`GET`	`/nodes/{nodeId}`	Obtener un nodo
`GET`	`/nodes/{nodeId}/children`	Listar los nodos hijos
`POST`	`/nodes/{nodeId}/children`	Crear un nodo
`PUT`	`/nodes/{nodeId}/content`	Actualizar el contenido de un nodo
`GET`	`/nodes/{nodeId}/content`	Obtener el contenido de un nodo

Tabla 1.6 Descripción del grupo *people* de la Core API

Método	URI	Acción
`GET`	`/people`	Lista todas las personas
`POST`	`/people`	Crea una nueva persona
`GET`	`/people/{personId}`	Obtiene una persona
`GET`	`/people/{personId}/avatar`	Obtiene el avatar de una persona

Tabla 1.7 Descripción del grupo *queries* de la Core API

Método	URI	Acción
`GET`	`/queries/nodes`	Obtiene una lista de nodos que coinciden con los criterios de búsqueda especificados

API Authentication: Permite iniciar y terminar sesión de un usuario en Alfresco con sus credenciales o mediante un *ticket*. Un *ticket* es un identificativo para mantener la sesión iniciada para un determinado *host*. La Tabla 1.8 muestra algunas de las acciones disponibles de esta API. Un *ticket* puede viajar sobre cualquier solicitud HTTP como cadena de consulta sobre la URL de la solicitud para proveer acceso a cualquier otro *end-point* de las API.

Tabla 1.8 Algunas acciones de la API Authentication

Método	URI	Acción
`POST`	`/tickets`	Crea un *ticket* (login)
`DELETE`	`/tickets/-me-`	Elimina un *ticket* (logout)
`GET`	`/tickets/-me-`	Valida un *ticket*

API CMN: Esta API no ha sido lanzada oficialmente, se dice que es de dominio privado, sin embargo existe mucha documentación sobre esta en [21]. La principal función de esta API es gestionar Modelos personalizados, es decir es la encargada de crear, modificar, eliminar y presentar Modelos, Tipos, Aspectos, Propiedades e incluso Constraints[32] de Propiedades, a su vez es posible activar o desactivar un Modelo. El cliente por defecto de Alfresco utiliza esta API para su gestión de Modelos de Contenido para administradores. En las solicitudes que genera y las respuestas que recibe esta API se maneja contenido JSON y *octect-stream*[33]. A diferencia de las API públicas que manejan JSON como estándar, esta API al no ser pública aún tiene partes que manejan contenido diferente a JSON.

Tabla 1.9 Acciones de la API ContentCMN

Método	URI Relativa	Acción
`POST`	`/cmm`	Crea un Modelo personalizado
`POST`	`/cmm/{modelName}/types`	Crea un Tipo personalizado
`POST`	`/cmm/{modelName}/aspects`	Crea un Aspecto personalizado
`GET`	`/cmm`	Obtiene todos los Modelos

[32] Constraints: Representa una restricción de la Propiedad. Puede representar una restricción de longitud, de valores permitidos, de lista de valores o incluso hasta restricciones del tipo expresiones regulares.

[33] `Application/octect-stream`: Representa datos binarios arbitrarios. Puede ser textos o archivos.

Tabla 1.8 Acciones de la API ContentCMN (Continuación)

Método	URI Relativa	Acción
POST	/cmm/{modelName}/types/{typeName}/constraints	Crea un Constraint personalizado
PUT	/cmm/{modelName}?select=status	Activa un Modelo personalizado
PUT	/cmm/{modelName}?select=status	Desactiva un Modelo personalizado
PUT	/cmm/{modelName}/aspects/{aspectName}?select=props	Añade una Propiedad a un Aspecto existente
PUT	/cmm/{modelName}/types/{typeName}?select=props	Añade una Propiedad a un Tipo existente
PUT	/cmm/{modelName}	Actualiza un Modelo personalizado
PUT	/cmm/{modelName}/types/{typeName}	Actualiza un Tipo personalizado
PUT	/cmm/{modelName}/aspects/{aspectName}	Actualiza un Aspecto personalizado
GET	/cmm/{modelName}	Obtiene un solo Modelo personalizado
GET	/cmm/{modelName}/types	Obtiene todos los Tipos personalizados
GET	/cmm/{modelName}/types/{typeName}	Obtiene un Tipo personalizado
GET	/cmm/{modelName}/aspects	Obtiene todos los Aspectos personalizados
GET	/cmm/{modelName}/aspects/{aspectName}	Obtiene un Aspecto personalizado
GET	/cmm/{modelName}/constraints	Obtiene todos los Constraints
GET	/cmm/{modelName}/constraints/{constraintName}	Obtiene un Constraint personalizado
DELETE	/cmm/{modelName}	Elimina un Modelo personalizado
DELETE	/cmm/{modelName}/types/{typeName}	Elimina un Tipo personalizado
DELETE	/cmm/{modelName}/aspects/{aspectName}	Elimina un Aspecto personalizado

API Content Search: A pesar de que esta API contiene solo un *end-point,* la lógica que lleva el cuerpo de las solicitudes permite realizar búsquedas al estilo SQL.

Tabla 1.10 Acciones de la API Content Search

Método	URI Relativa	Acción
`POST`	`/search`	Búsqueda en Alfresco

Swagger: Swagger es un *framework* de código abierto que permite diseñar y describir la estructura de una API de tal manera que sea posible documentar, probar y consumir una API. Swagger es un enlace tanto para el dueño de la API como para quien desee consumirla, este enlace es logrado a través de archivos de extensión YAML[34] o JSON que contienen una detallada descripción de lo que es la API, es decir, que operaciones soporta, los parámetros y retornos, autorizaciones, información de contacto e incluso licencias. Estos archivos además permiten autogenerar librerías cliente para diferentes lenguajes de programación que sirvan como modelos (clases) y como operaciones (interfaces), siguiendo la especificación OpenAPI[35] [22].

1.3.8 Scrum

Scrum es una metodología ágil de desarrollo de *software*. Sin embargo, más que una metodología rígida definitiva, Scrum puede ser visto como un *framework* del cual se puede tomar procesos para aplicarlos al desarrollo de un producto [23]. Scrum se basa en la mejora continua del *software* mientras se lo está desarrollando. Scrum cuenta con tres roles principales (ver Figura 1.19) Dueño del Producto, Maestro Scrum y el Equipo Scrum.

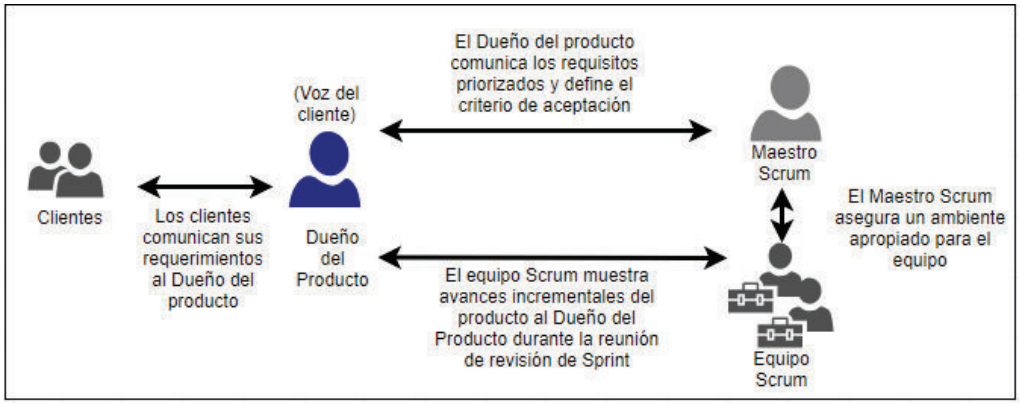


Figura 1.19 Roles Scrum [23]

[34] YAML: Lenguaje de serialización de datos legible al ser humano, basado en XML y en Python.
[35] OpenAPI: Especificación neutral de vendedor, portable y abierta para descripción de API.

Fases Scrum: Scrum está organizado en cuatro fases: Inicio, Planeación, Diseño e Implementación, y Revisión y Retrospección, resumidas en la Tabla 1.11. La fase de Inicio es la única que no se itera, las demás fases son iterativas.

Tabla 1.11 Fases de Scrum

Fase	Procesos	Descripción
Inicio	Crear la visión del proyecto	En esta fase se plantea la visión del proyecto. Esta visión sirve de inspiración y provee enfoque a todo el proyecto, en esta etapa se identifica al Dueño del producto.
	Identificar al Maestro Scrum y *stakeholders*	Se identifica al Maestro Scrum y *stakeholders*.
	Desarrollar *Epics*	Un *Epic* puede ser definido como pedazo grande de trabajo que tiene un objetivo en común. Se podría decir que un *Epic* es una historia de usuario definida en términos generales.
Planeación	Crear las historias de usuario	A partir de cada *Epic* se desglosan historias de usuario detalladas y escritas por el Dueño del Producto. Estas historias de usuario son simples, cortas y fáciles de implementar.
	Crear Tareas	Las historias de usuario aprobadas son desglosadas en tareas específicas y reagrupadas en una lista de tareas.
	Crear el s*print backlog*	El s*print backlog* son grupos de tareas a realizarse en un s*print*. Aquí se utilizan tableros Scrum.
Diseño e Implementación	Crear entregables	Se trabaja sobre algún s*print* del s*print backlog* para generar entregables. Los tableros Scrum son utilizados para hacer un seguimiento del trabajo y las actividades realizadas.
	Actualizar y mantener el *backlog*.	Cualquier cambio o actualización del *backlog* del producto es discutido e incorporado en esta etapa.
Revisión y Retrospección	Mostrar y validar los *sprints*	Los entregables de cada *sprint* son revisados por el Dueño del Producto y/o *stakeholders.* La idea es asegurar aprobación y aceptación del producto.
	Retrospección de *sprints*	El Equipo Scrum y el Maestro Scrum discuten las lecciones aprendidas en cada *sprint*.
	Retrospección del proyecto	El dueño del producto ayuda a identificar y documentar las lecciones aprendidas de todo el proyecto.

Tablero Scrum: Es un tablero de tareas que muestra el progreso del equipo en cada *sprint*. Cuenta con cuatro columnas, una para cada estado de una tarea de un determinado *sprint*. En un inicio todas las tareas van por defecto a la columna 'Por Hacer'. Una vez que una

tarea se ha empezado a hacer esta pasa el estado ‘En Progreso’. Adicionalmente existen los estados ‘Probando’ y ‘Realizada’. La Figura 1.20 muestra un ejemplo de tablero Scrum.

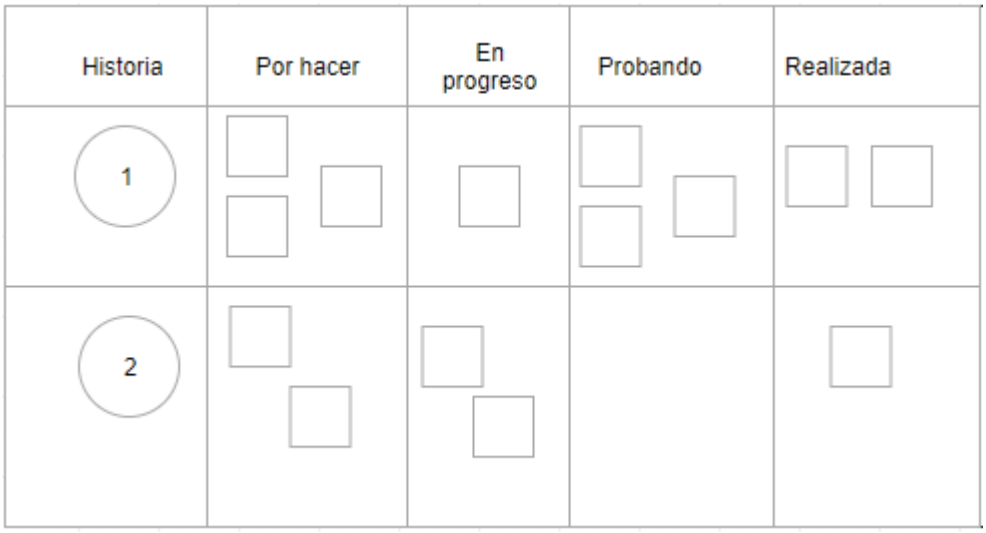


Figura 1.20 Tablero Scrum

Levantamiento de información: Aunque las historias de usuario pueden ser escritas directamente por el dueño del producto, en el caso de que este esté al tanto del giro del negocio y de los intereses del cliente, existen otras herramientas que permiten levantar información faltante. Para capturar historias de usuario es posible crear reuniones (planificadas y moderadas) con grupos de usuarios, realizar entrevistas o utilizar cuestionarios. Sin embargo, la idea iterativa de Scrum hace que el enfoque para el levantamiento de información esté iteraciones de *software* y no en actas de reunión o encuestas realizadas.

1.3.9 Herramientas compatibles con .NET utilizadas para el desarrollo

Autorest: Autorest genera librerías cliente para acceder a servicios web RESTful a través de un archivo descriptor de API REST que use el formato de especificación OpenAPI. Autorest es gratuito y multiplataforma, también forma parte de las herramientas disponibles de Microsoft Visual Studio Enterprise 2017.

Para hacer uso de Autorest en Microsoft Visual Studio Enterprise 2017, basta con agregar, mediante botón derecho, un cliente de API REST a un proyecto y luego seleccionar el archivo descriptor de API en formato JSON, como lo muestra la Figura 1.21. El resultado (Figura 1.22) será una estructura de carpetas con archivos de clase que muestran las clases asociadas a las entidades de la API y reflejan el comportamiento de la API a través de interfaces.

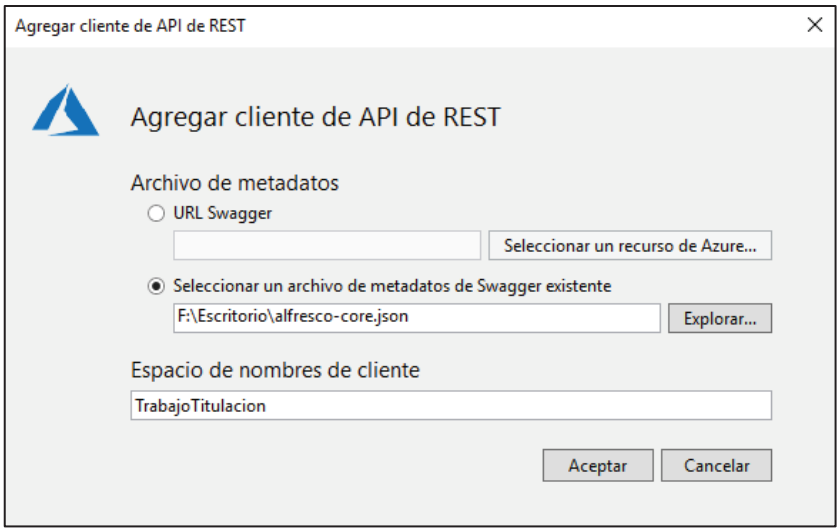


Figura 1.21 Utilización de Autorest desde MVSE 2017

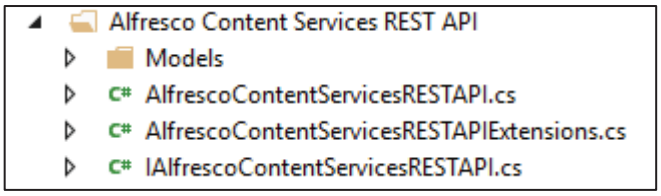


Figura 1.22 Contenido autogenerado utilizando Autorest

Netwonsoft: Newtosoft o Json.NET es un paquete de código abierto serializador/deserializador de alto rendimiento para interactuar con contenido JSON desde C# [24]. Sus dependencias pueden ser añadidas a un proyecto a través de paquetes NuGet[36]. Newtonsoft permite asociar objetos C# a objetos JSON y viceversa, para ello se debe proporcionar una estructura de decoradores a los atributos que se van a serializar/deserializar o simplemente trabajar con objetos dinámicos. Un ejemplo de uso de Json.NET, donde se serializa un objeto `Product` y se muestra en comentarios su impresión en pantalla, se presenta en la Figura 1.23; en la parte (a) está definida la clase `Product` (las etiquetas JSON definen que se serializa/deserializa y que no) y en la parte (b) se muestra la serialización de un objeto `Product`, el `Id` no se serializa debido a que tiene su etiqueta de serialización/deserialización es la etiqueta `[JsonIgnore]`.

RestSharp: RestSharp es un API local que proporciona acceso a llamadas HTTP enfocándose en llamadas sobre API REST. RestSharp provee una estructura de métodos sobre la clase `RestClient` que permite agregar parámetros, segmentos URL, cabeceras,

[36] NuGet: Gestor de dependencias diseñado para la plataforma de desarrollo de Microsoft. Añade automáticamente y enlaza ensamblados `.dll` a proyectos C#.

archivos o cadenas de consulta a una solicitud HTTP. Además, RestSharp provee una estructura para ejecutar llamadas asíncronas. En la Figura 1.24 se muestra la creación de una solicitud síncrona HTTP a la URI `http://twitter.com` con credenciales `username` y `password` y a la espera del recurso `statuses/friends_timeline.xml`.

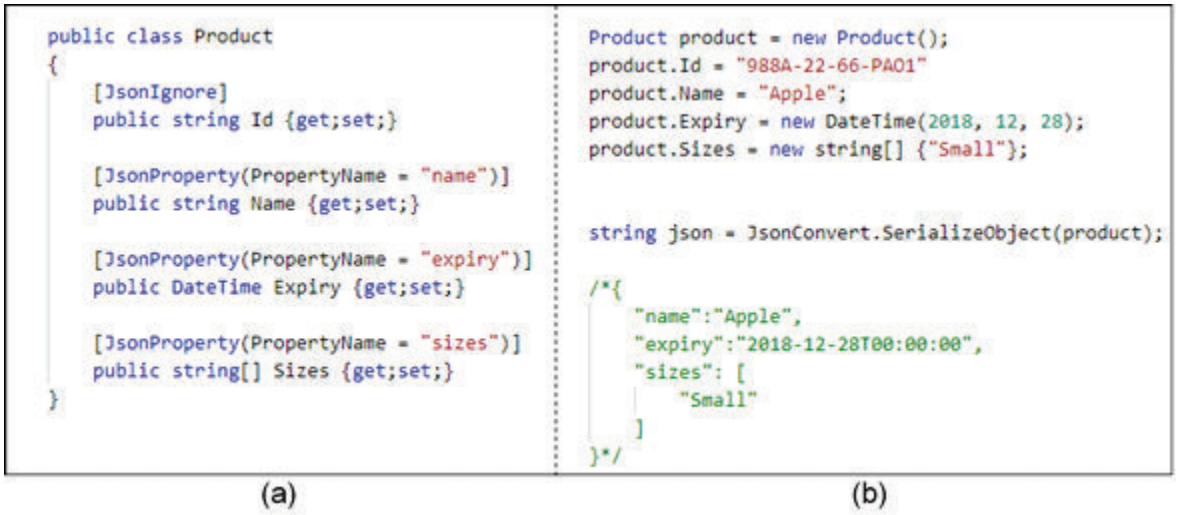


Figura 1.23 Ejemplo de serialización a través de Json.NET

```
using RestSharp;
using RestSharp.Authenticators;

var client = new RestClient();
client.BaseUrl = new Uri("http://twitter.com");
client.Authenticator = new HttpBasicAuthenticator("username", "password");

var request = new RestRequest();
request.Resource = "statuses/friends_timeline.xml";

IRestResponse response = client.Execute(request);
```

Figura 1.24 Ejemplo de uso de RestSharp [25]

DSO File (Microsoft Developer Support OLE Files)*:* Es un ensamblado `.dll` que, entre otras cosas, permite la creación, lectura, eliminación y modificación de propiedades asociados a la mayoría de archivos y carpetas del sistema operativo Windows. Los archivos que soportan la edición de propiedades mediante DSO File deben estar basados en la estructura de almacenamiento OLE[37] (*Object Linking and Embedding*), entre las extensiones compatibles están: `.doc`, `.dot`, `.xls`, `.xlt`, `.ppt`, `.pps`, `.pdf`, `.rar`,

[37] OLE (*Object Linking and Embedding*): Tecnología de Microsoft para importar y exportar parte de un archivo/documento a otra aplicación.

`.txt`, carpetas Windows, etc. Requiere instalación [26]. En la Figura 1.25 se muestra la adición, desde C#, de la propiedad de nombre "`Nombre Propiedad`" y de valor "`Valor Propiedad`" al documento `prueba.pdf` ubicado en `F:\Documents`.

```
string filePath = @"F:\Documents\prueba.pdf";
DSOFile.OleDocumentProperties myFile = new DSOFile.OleDocumentProperties();
myFile.Open(filePath);
myFile.CustomProperties.Add("Nombre Propiedad", "Valor Propiedad");
myFile.Save();
myFile.Close(true);
```

Figura 1.25 Ejemplo de uso de `DSOFile`

Document Format Open XML*:* Es un paquete que ofrece herramientas para trabajar con archivos del tipo Word, Excel y Power Point, es decir archivos con extensiones: `.docx`, `.docm`, `.dotx`, `dotm`, `.xlsx`, `.xlsm`, `.xltx`, `.xltm`, `.xlsb`, `.xlam`, `.pptx`, `.pptm`, `.ppsx`, `.ppsm`, etc posterior a la versión 2007 [27]. Entre otras cosas, permite la creación, lectura, eliminación y modificación de propiedades personalizadas de archivos. La Figura 1.26 muestra la constancia de creación de una propiedad con Open XML desde C#. La propiedad es de nombre `IdAlfresco`, el valor es `a3ce908b-f…` y de tipo Texto.

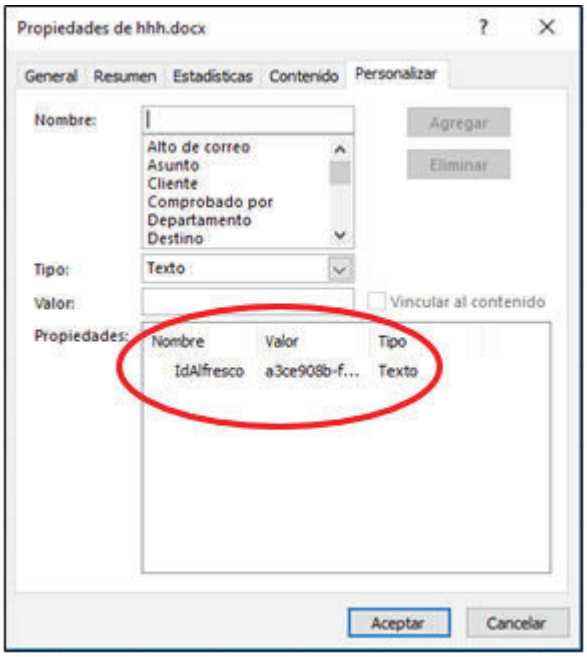


Figura 1.26 Propiedad personalizada creada a través de Open XML

Llamadas asíncronas: C# desde su versión 5, ofrece un esquema que facilita el uso de métodos asíncronos a través de las palabras claves `async` y `await`.

`async` se aplica a la firma de un método después del modificador de acceso, e indica que tal método esperará a que los métodos asíncronos que se ejecutan dentro de este método, finalicen [28].

`await` se aplica en la llamada de un método asíncrono y permite que tal llamada sea esperada por el método en donde se realiza la llamada, `async` va de la mano de `await`. Si se desea que un método asíncrono sea esperado hasta el final de su ejecución (sin bloquear el hilo principal de ejecución) entonces se utiliza la palabra `await` en la llamada, pero esto implica que el método de donde se llama contenga la palabra `async`.

`Task` es otra palabra clave que se utiliza dentro de la estructura asíncrona. Representa a una promesa de retorno, por ejemplo, `Task<string>` (escrito en un tipo de retorno de un método) representa un tipo de retorno `string` que se retornará cuando esté listo y no necesariamente en el momento de su llamada. Técnicamente `Task<T>` es una clase perteneciente a la librería Task Parallel Library y representa una operación asíncrona [29]. Un objeto del tipo `Task` puede ser utilizado para correr operaciones en paralelo.

A través de estas llamadas asíncronas se evita el bloqueo de la interfaz gráfica mientras se ejecuta una llamada remota.

En la Figura 1.27 se muestra un ejemplo del uso de las palabras clave `async`/`await`. La ejecución en consola del programa ejemplo producirá un mensaje inicial y empezará a contar el número de caracteres de una página web en segundo plano (asíncronamente) sin bloquear el programa, una vez que la respuesta esté lista aparecerá en pantalla los mensajes de finalización. El método `Main` contiene la secuencia inicial del programa y deberá ser asíncrono (`async`, debido a que contiene una llamada asíncrona) y con tipo de retorno `Task` para permitir que `Main` espere (sin bloqueo) al método `Contador` que se llama desde `Main`. El método `Contador` deberá esperar a que el método `GetPageLengthsAsync` termine para imprimir el resultado de la cuenta y un mensaje de finalización. El método `GetPageLengthsAsync` tiene un tipo de retorno `Task<long>` lo que significa que retornará asíncronamente un tipo `long`. Además, `GetPageLengthsAsync` es quién ejecuta la llamada remota HTTP y deberá esperar a que el recurso solicitado sea respondido.

Esta secuencia en cascada de métodos asíncronos indica que los métodos se ejecutarán secuencialmente esperando cada llamada asíncrona que se realice. una vez que todas las llamadas han sido completadas, entonces la ejecución volverá a su curso normal, la ventaja

es que nunca se bloqueó el programa y si se habla de una interfaz gráfica entonces la interfaz será la que no se bloquee mientras se espera el retorno de la llamada asíncrona.

```
static async Task Main()
{
    Console.WriteLine("Estamos contando el número de páginas y caracteres");
    await Contador();
    Console.WriteLine("Good bye");
    Console.ReadLine();
}

1 referencia
private static async Task Contador()
{
    string[] uris = { "https://blogs.msdn.microsoft.com/benwilli/2017/12/08/" +
            "async-main-is-available-but-hidden/"};
    long characters = await GetPageLengthsAsync(uris);
    Console.WriteLine($"{uris.Length} pages, {characters:N0} characters");
    Console.WriteLine("Ha finalizado");
}

1 referencia
private static async Task<long> GetPageLengthsAsync(string[] uris)
{
    var client = new HttpClient();
    long pageLengths = 0;

    foreach (var uri in uris)
    {
        var escapedUri = new Uri(Uri.EscapeUriString(uri));
        string pageContents = await client.GetStringAsync(escapedUri);
        Interlocked.Add(ref pageLengths, pageContents.Length);
    }
    return pageLengths;
}
```

Figura 1.27 Ejemplo del uso de `async/await` [30]

Team: La característica Team de Visual Studio Enterprise 2017 permite controlar cambios en el código de un proyecto. Es una herramienta colaborativa basada en el estilo Git[38] que permite versionar el código mientras se lo va desarrollando, esto posibilita tener un respaldo del código del prototipo, por versiones e identificando el autor de esos cambios.

Además, Team permite mantener un repositorio versionado localmente y también un repositorio en la nube, como por ejemplo en GitHub[39] [31]. Para ello es necesario sincronizar una cuenta de GitHub a Team y automáticamente todos los *pushs*[40] se harán de manera local y remota. En la Figura 1.28 se muestra la barra de herramientas del Team, ubicada por defecto en la parte inferior derecha de Visual Studio. Esta barra de opciones

[38] Git: Sistema de control de versiones de código.
[39] GitHub: plataforma de desarrollo colaborativo para almacenar proyectos en la nube utilizando Git.
[40] *Push*: Actualización justificada del código en una rama de versión del mismo.

permite subir los cambios aprobados (flecha hacia arriba), aprobar los cambios (lápiz), conocer la ruta de acceso del proyecto e insertar nuevas ramas de ser necesario.

En la Figura 1.29 se muestra el explorador del Team a través de la cual es posible realizar comentarios sobre los cambios realizados, comentarios sobre las actualizaciones (cambios aprobados), sincronizaciones de entrada o de salida y configuraciones de repositorio remoto.

Figura 1.28 Barra de Herramientas de Team

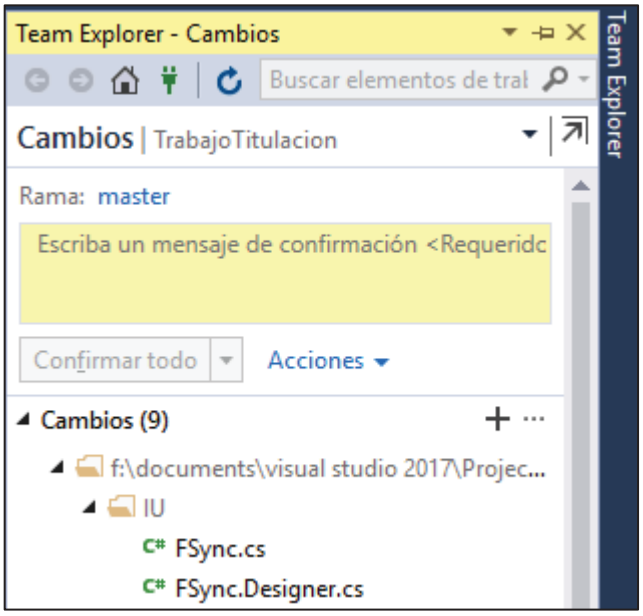


Figura 1.29 Explorador del Team

1.4 Relación del estudio con trabajos anteriores del área

Anteriormente se ha realizado el trabajo de [32], el cual presenta el "Desarrollo e implementación de un prototipo de seguridad web para la gestión y administración de documentos para el sistema Alfresco". En el proyecto anteriormente mencionado se implementó una aplicación web que permite gestionar documentación mediante Alfresco. La aplicación web consiste en un cliente para el VIPS (Vicerrectorado de Investigación y Proyección Social) que utiliza Alfresco como *back-end*. También, se utilizan modelos de datos (metadatos), se realiza control de acceso mediante roles y se realiza acceso y modificación de documentos gestionados por Alfresco mediante el estándar CMIS (*Content Management Interoperability Services*).

El presente Trabajo de Titulación se enfoca en la administración de contenidos de Alfresco de una forma más amplia ya que esta implementación no se trata de una particularización o integración con algún sistema propietario, sino que este prototipo funciona para cualquier usuario de Alfresco. Además, en el presente proyecto se implementarán cuatro actividades puntuales que no se realizaron en el proyecto afín, estas actividades son:

a. Creación de metadatos de acuerdo con la necesidad actual del usuario.

b. Carga masiva de contenidos.

c. Sincronización de contenido fuera de línea.

d. Búsqueda de contenidos basada en metadatos.

Además, la aplicación implementada en el proyecto afín es del tipo web, mientras que la aplicación que se desarrollará en este Trabajo de Titulación es del tipo escritorio.

Para la comunicación cliente-servidor, en el Trabajo de Titulación que se realizará, se hará uso de las API REST de Alfresco, mientras que en el proyecto de titulación comparado se hace uso del estándar CMIS.

2 METODOLOGÍA

Para la consecución de este Trabajo de Titulación se realizó una investigación aplicada y se empleó la metodología ágil de desarrollo de *software* Scrum. Esta metodología plantea una fase inicial donde se desarrolla la visión del proyecto, los roles Scrum y algunos *epics*. Luego, se plantea la fase de Planeación donde se ha realizado el levantamiento de información e identificación de *sprints*. Después, en la fase de Diseño e Implementación se ha realizado un incremento del producto, potencialmente entregable, por cada *sprint*. La fase de Revisión y Retrospectiva permiten iterar cada incremento hasta que este sea aceptado por el usuario.

En la fase inicial se planteó la visión del proyecto, esta es desarrollar un prototipo de aplicación distribuida que permita gestionar los contenidos disponibles en Alfresco. Se identificaron cuatro *epics* dados por el dueño del producto.

En la fase de planeación se procedió a escribir las historias de usuario, identificar los requerimientos y dividir los mismos en pequeñas tareas. Posterior a ello se agrupó requerimientos y sus tareas por realizar para formar *sprints.*

En la fase de diseño e implementación se fue trabajando sobre cada *sprint*. En cada *sprint* se generó un bosquejo o *sketch* de la interfaz de usuario y se determinó el flujo de interacción del usuario con la aplicación a través de *wireframes*. Luego de ello, se procedió a generar los diagramas arquitectónicos de las interfaces de comunicación cliente-servidor y de las clases que reflejan el dominio de la aplicación, para cumplir con cada requerimiento. Consiguiente a ello, se procedió a generar el incremento del producto (es decir a escribir en lenguaje C# cada interfaz de usuario, cada diagrama de clases UML y cada algoritmo). En esta fase también se actualizó el *sprint backlog* es decir si mientras se diseñaba o implementaba se identificó un nuevo requerimiento, este es agregado a la lista de requerimientos dependiendo del alcance de este Trabajo de Titulación.

En la fase de Revisión y Retrospectiva el dueño del producto es el encargado de validar los requerimientos de usuario y solicitar cambios para asegurar aceptación de cada entregable. En esta fase se discutieron cambios en la arquitectura de la aplicación y en la interfaz de usuario. Estos cambios fueron registrados y se muestran en este documento, al final de la explicación de cada *sprint.*

La documentación de cada entregable se la ha realizado por cada *sprint*. Cada *sprint* muestra las fases de Diseño e Implementación y Revisión y Retrospectiva con el objetivo de mostrar cómo se fue realizando la iteración de cada entregable.

2.1 Levantamiento de información

Para levantar las primeras historias de usuario fue necesario identificar los roles Scrum. Los roles Scrum se identifican en la Tabla 2.1.

Tabla 2.1 Roles Scrum

Rol	**Nombre**
Maestro Scrum	David Mejía
Dueño del Producto	David Mejía
Desarrollador	Josue Burbano

El dueño del producto es quien se encargó de identificar cuatro *epics*, los cuales fueron la base del levantamiento de información, estos son*:*

1. Verificación de cambios en los contenidos del prototipo y actualización de tales cambios sobre los contenidos símiles del servidor, y viceversa.
2. Creación y adición de metadatos a los contenidos.
3. Carga masiva de contenidos a través de la opción arrastrar y soltar.
4. Búsqueda basada en metadatos.

2.1.1 Encuestas

A partir de los *epics* se realiza un diseño de encuesta para validar la información dada en los *epics*. La encuesta se muestra en el Anexo I.

Los resultados de la encuesta se detallan a continuación. En la Figura 2.1 se muestran los resultados de la pregunta: ¿de qué manera accede a Alfresco?, esta pregunta se la realizó con el fin de identificar si existe otro cliente de Alfresco que realice funcionalidades adicionales al cliente por defecto; los resultados de esta pregunta indican que el 100% de los encuestados acceden a Alfresco a través del cliente por defecto, es decir a través de Alfresco Share.

En la Figura 2.2 se muestran los resultados de la pregunta: ¿cree conveniente tener la posibilidad de sincronizar sus archivos en el computador con Alfresco al estilo Google Drive?, esta pregunta se realizó con el fin de validar el primer *epic*. Este *epic* quedó totalmente validado ya que el 100% de los encuestados considera conveniente sincronizar archivos en el computador con Alfresco al estilo del cliente de Google Drive. En una encuesta se menciona que existe otro cliente, el cual permite realizar sincronización y que

es compatible con Alfresco. Se trata del *software* CmisSync[41], el cual permite sincronizar Alfresco al estilo del cliente de Google Drive.

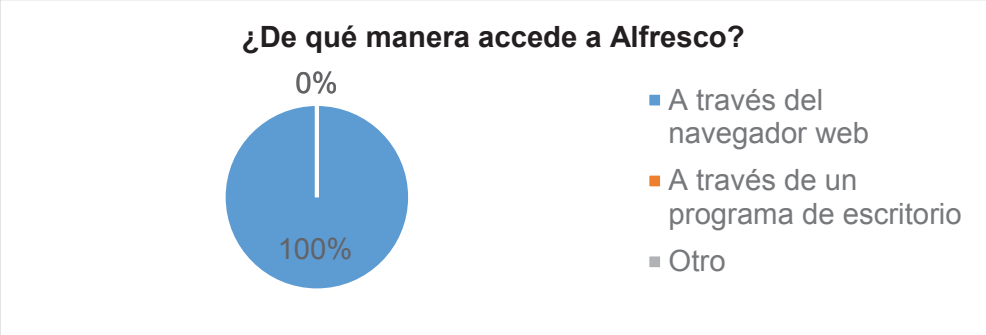


Figura 2.1 Resultados de la pregunta 1

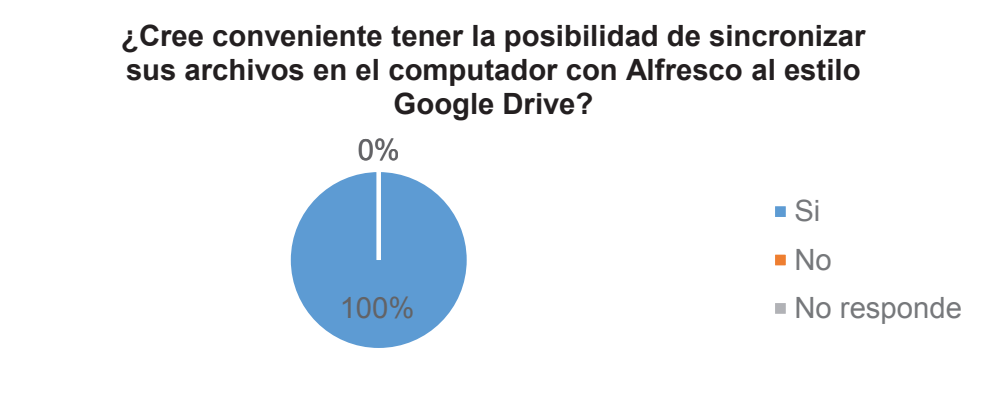


Figura 2.2 Resultados de la pregunta 2

En la Figura 2.3 se muestran los resultados de la pregunta: ¿cuándo realiza la carga de contenido en Alfresco, existe la posibilidad de subir archivos mediante arrastrar y soltar?, esta pregunta se la realizó con la intención de conocer si la acción de arrastrar y soltar estaba implementada en el cliente Alfresco por defecto. Los resultados dieron a conocer que el 80% de los encuestados reconocen que existe la posibilidad de arrastrar y soltar archivos, el 10% restante consideran que esta funcionalidad no está implementada y el 10% no contestó.

En la Figura 2.4 se muestran los resultados de la pregunta: cree usted que se facilitaría la carga de contenido a Alfresco, ¿si existe la posibilidad de arrastrar y soltar archivos?, esta pregunta fue realizada con el objetivo de implementar la funcionalidad de arrastrar y soltar sobre el prototipo de la aplicación. La aceptación de esta propuesta fue del 90% mientras que el 10% del total de los encuestados no respondió esta pregunta.

[41] CmisSync: permite sincronizar un servidor de ECM compatible con CMIS con la PC local [40].

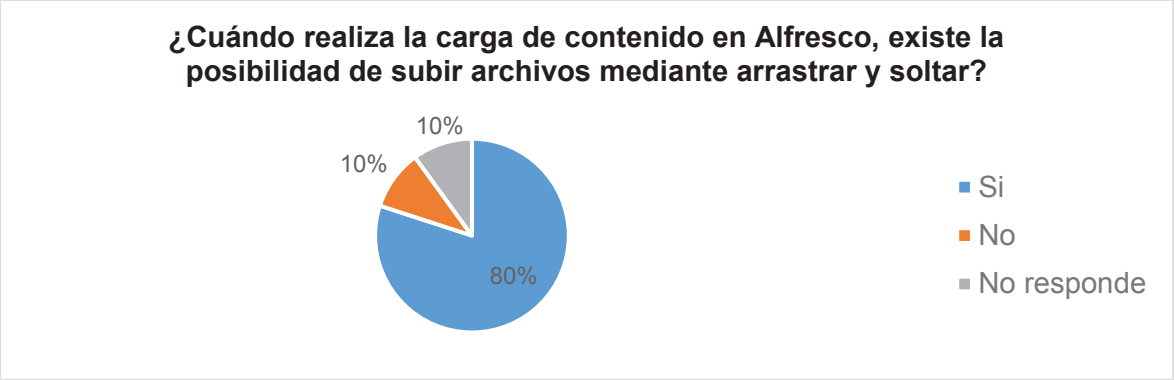


Figura 2.3 Resultados de la pregunta 3

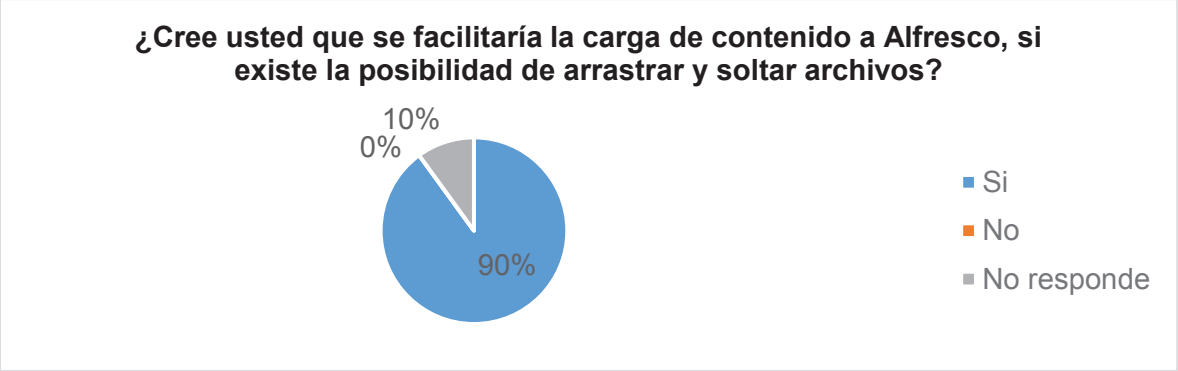


Figura 2.4 Resultados de la pregunta 4

En la Figura 2.5 se muestran los resultados de la pregunta: ¿es posible cargar contenido masivamente (muchos archivos a la vez)? El objetivo de esta pregunta fue validar el tercer *epic*. El 55% de los encuestados consideró que sí es posible subir contenido masivamente, mientras que el 45% consideró que no es posible.

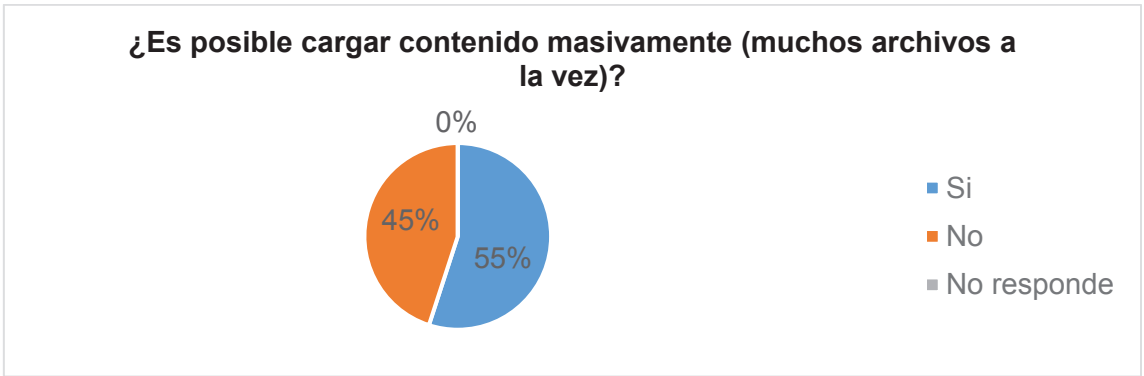


Figura 2.5 Resultados de la pregunta 5

En la Figura 2.6 se muestran los resultados de la pregunta: ¿necesita cargar archivos masivamente?, esta pregunta se la realizó con el objetivo de validar el tercer *epic*. El 100% de los encuestados está de acuerdo en la necesidad de cargar archivos masivamente.

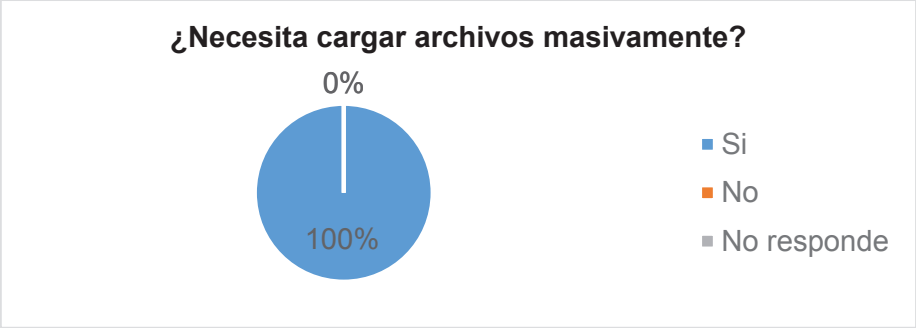


Figura 2.6 Resultados de la pregunta 6

En la Figura 2.7 se muestran los resultados de la pregunta: ¿la búsqueda de contenidos en Alfresco resulta complicada?, esta pregunta fue realizada con el objetivo de rediseñar la opción de búsqueda del cliente por defecto de Alfresco. El 55% de los encuestados están de acuerdo con que la búsqueda de contenidos resulta complicada.

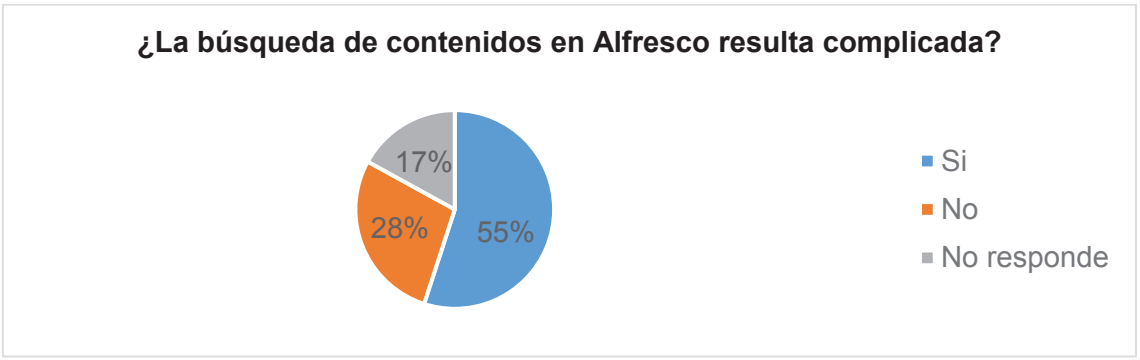


Figura 2.7 Resultados de la pregunta 7

En la Figura 2.8 se muestran los resultados de la pregunta: ¿sería útil poder agregar metadatos personalizados (características de los contenidos) para efectivizar las búsquedas de contenidos?, esta pregunta se la realizó con el fin de validar el segundo y cuarto *epic*. Los resultados mostraron un total apoyo a la iniciativa.

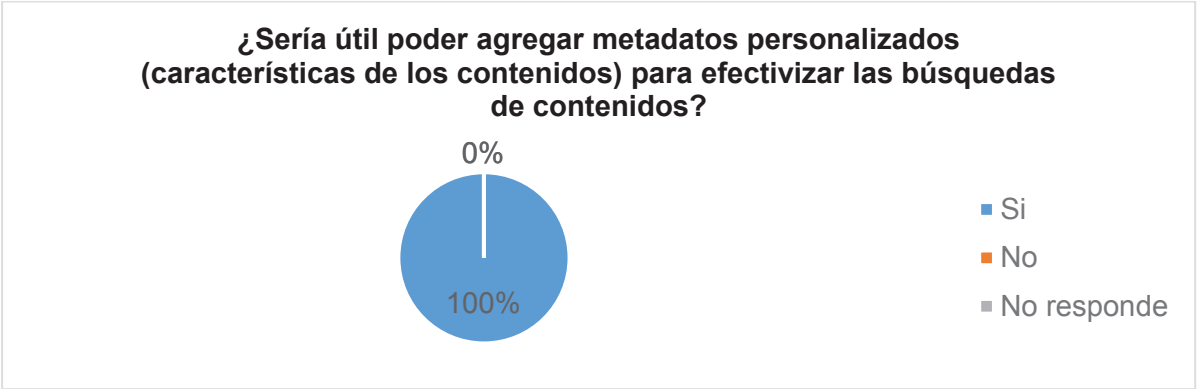


Figura 2.8 Resultados de la pregunta 8

La pregunta 9: ¿ha tenido algún problema con Alfresco?, y la pregunta 10: ¿qué mejoras le aplicaría a Alfresco? son preguntas abiertas y permiten identificar funcionalidades faltantes o por mejorar del cliente por defecto de Alfresco. En la Tabla 2.2 se presentan los resultados.

Tabla 2.2 Problemas con Alfresco

Dificultad	**Frecuencia**
Lentitud y servidor fuera de línea	4
Arrastra y soltar carpetas a la vez	4
Mover archivos dentro de Alfresco con facilidad	2
Estructura simplificada de directorios	2
Integración con otros sistemas de la EPN	1
Facilidad para editar nombres de los archivos subidos	1
Mejorar seguridades	1

2.1.2 Historias de usuario

Los resultados de las encuestas iniciales permitieron organizar la información y así crear historias de usuario. Las historias de usuario nacen de una selección de las necesidades de los usuarios (identificadas en las encuestas) que se alinean con el alcance de este Trabajo de Titulación. Las historias de usuario se presentan en la Tabla 2.3.

Tabla 2.3 Historias de usuario

ID	**Título**	**Descripción**
HU01	Verificación de cambios y actualización de los mismos	El usuario necesita sincronizar sus archivos con Alfresco al estilo del cliente de Google Drive.
HU02	Arrastrar y soltar	El usuario necesita cargar archivos y carpetas a Alfresco mediante arrastrar y soltar
HU03	Carga masiva de carpetas	El usuario necesita cargar archivos y carpetas a Alfresco masivamente
HU04	Agregar metadatos personalizados	El usuario necesita agregar metadatos personalizados
HU05	Búsqueda basada en metadatos	El usuario necesita buscar archivos o carpetas a través de metadatos personalizados
HU06	Mover archivos sobre el repositorio de usuario	El usuario busca facilidad para mover archivos de una carpeta a otra dentro del repositorio de usuario
HU07	Cambiar nombre de archivos y carpetas	El usuario necesita facilidad para cambiar el nombre de archivos y carpetas
HU08	Estructura simplificada de directorios	El usuario necesita simplicidad en la estructura de directorios sobre los cuales sube su contenido

2.1.3 Requerimientos

A partir de las historias de usuario se identificaron requerimientos del prototipo. Estos requerimientos nacen del refinamiento de las historias de usuario desde un punto de vista técnico y desde la perspectiva del sistema. Los requerimientos están limitados por el alcance. Además, aquí se presentan todos los requerimientos, es decir los que se encontraron mientras se desarrolla, y no solamente los iniciales.

1. Iniciar sesión con las credenciales de Alfresco.
2. Cerrar sesión.
3. Acceder a un formulario que sirva de acceso a los diferentes módulos.
4. Permitir la visualización de archivos y carpetas de usuario.
5. Permitir cargar masivamente archivos y carpetas a través de arrastrar y soltar.
6. Permitir leer y editar metadatos de archivos y carpetas.
7. Permitir la creación, lectura, edición y eliminación de Modelos de Contenido.
8. Permitir la activación y desactivación de Modelos de Contenido.
9. Permitir la creación, lectura, edición y eliminación de Tipos.
10. Permitir la creación, lectura, edición y eliminación de Aspectos.
11. Permitir la creación, lectura, edición y eliminación de Propiedades.
12. Agregar y almacenar de sub-repositorios de Alfresco en la PC.
13. Permitir la sincronización de los contenidos de Alfresco con la PC.
14. Permitir la búsqueda de contenidos basada en tipos, aspectos, propiedades y valores de las propiedades.

2.1.4 Tareas

Cada requerimiento se lo ha desglosado en tareas factibles, simples y concretas. La Tabla 2.4 presenta las tareas a realizar por cada requerimiento, en la cual la columna Estado representa su estado (Por hacer, En proceso, Probando, Realizada) en el tablero Scrum. Inicialmente, todas las tareas tienen el estado “Por hacer”. Conforme se fue implementando cada tarea, esta fue cambiando de estado a “En progreso”, “Probando” y “Realizada”, ese cambio fue reflejado en el tablero Scrum.

Tabla 2.4 Desglosamiento de los requerimientos en tareas (Parte 1 de 2)

Requerimiento	Tareas	Estado
1. Iniciar sesión con las credenciales de Alfresco	1. Autenticar las credenciales de usuario contra el servidor de Alfresco	Por hacer
	2. Identificar al usuario autenticado mostrando el nombre del mismo	Por hacer
	3. Crear un Ticket de usuario que permita la identificación posterior al inicio de sesión	Por hacer
2. Cerrar sesión	4. Eliminar el Ticket de usuario local y salir del prototipo	Por hacer
	5. Eliminar el Ticket de usuario remoto	Por hacer
3. Acceder a un formulario que sirva de acceso a los diferentes módulos	6. Crear un tablero de usuario a través del cual sea posible acceder a los diferentes módulos del prototipo	Por hacer
	7. Crear un menú de usuario a través del cual sea posible acceder a los diferentes módulos del prototipo	Por hacer
4. Permitir la visualización de archivos y carpetas de usuario	8. Crear un visualizador navegable de carpetas de usuario	Por hacer
	9. Construir un visualizador de archivos y sus propiedades principales	Por hacer
5. Permitir cargar masivamente archivos y carpetas a través de arrastrar y soltar	10. Permitir arrastrar y soltar archivos o carpetas para cargarlos remotamente	Por hacer
	11. Cargar carpetas con su contenido	Por hacer
6. Permitir leer y editar metadatos de archivos y carpetas	12. Cambiar el nombre de un archivo fácilmente	Por hacer
	13. Leer metadatos de un archivo o carpeta	Por hacer
	14. Editar metadatos de un archivo o carpeta	Por hacer
	15. Agregar metadatos personalizados a un archivo o carpeta	Por hacer
7. Permitir la creación, lectura, edición y eliminación de Modelos de Contenido	16. Crear Modelos de Contenido	Por hacer
	17. Leer los Modelos de Contenido del usuario	Por hacer
	18. Editar Modelos de Contenido	Por hacer
	19. Eliminar Modelos de Contenido	Por hacer
8. Permitir la activación y desactivación de Modelos de Contenido	20. Activar un Modelo de Contenido	Por hacer
	21. Desactivar un Modelo de Contenido	Por hacer
9. Permitir la creación, lectura, edición y eliminación de Tipos	22. Crear Tipos	Por hacer
	23. Leer los Tipos del usuario	Por hacer
	24. Editar Tipos	Por hacer
	25. Eliminar Tipos	Por hacer
10. Permitir la creación, lectura, edición y eliminación de Aspectos	26.Crear Aspectos	Por hacer
	27. Leer los Aspectos del usuario	Por hacer
	28. Editar Aspectos	Por hacer
	29. Eliminar Aspecto	Por hacer

Tabla 2.4 Desglosamiento de los requerimientos en tareas (Parte 2 de 2)

11. Permitir la creación, lectura, edición y eliminación de Propiedades	30. Crear Propiedades	Por hacer
	31. Leer las Propiedades de un Tipo o Aspecto del usuario	Por hacer
	32. Editar las Propiedades de un tipo o aspecto	Por hacer
	33. Leer las Propiedades de un tipo o aspecto del usuario	Por hacer
12. Agregar y almacenar sub-repositorios de Alfresco en la PC	34. Desplegar todo el repositorio de usuario	Por hacer
	35. Guardar y almacenar los contenidos de un sub-repositorio seleccionado	Por hacer
13. Permitir la sincronización de los contenidos de Alfresco con la PC	36. Actualización en contenidos de la PC sobre los contenidos símiles del servidor	Por hacer
	37. Actualización en contenidos de Alfresco sobre los contenidos símiles de la PC	Por hacer
14. Permitir la búsqueda de contenidos basada en tipos, aspectos, propiedades y valores de las propiedades	38. Buscar contenidos por Tipo	Por hacer
	39. Buscar contenidos por Aspecto	Por hacer
	40. Buscar contenidos por Propiedades	Por hacer
	41. Buscar contenidos por coincidencia del valor de la Propiedad	Por hacer
	42. Buscar contenidos por comparación lógica del valor de la Propiedad	Por hacer
	43. Buscar contenidos por comparación entre dos valores del valor de la Propiedad	Por hacer

2.1.5 Tablero Scrum

Para dar seguimiento a cada tarea de cada requerimiento se generó un tablero Scrum. En la Figura 2.9 se muestra la fase inicial del tablero Scrum con dos requerimientos (Requerimiento 3 y 8) y sus respectivas tareas. Este tablero Scrum fue implementado utilizado la herramienta *online* Taiga.io[42]. Las columnas *Ready*, *In Progress, Ready for Test* y *Done* fueron utilizadas correspondientemente a las columnas definidas en este proyecto para Scrum: Por hacer, En progreso, Probando y Realizada.

2.1.6 Sprint backlog

Los requerimientos agrupados forman *sprints*. El conjunto de *sprints* pasan a ser parte del *sprint backlog*. La agrupación de requerimientos se lo realiza tomando en cuenta que cada *sprint* deberá generar un entregable. El *sprint backlog* que posee todos los *sprints* y la relación entre *sprints* y requerimientos se muestra en la Tabla 2.5.

[42] Taiga.io: Plataforma gratuita y de código abierto para administración de proyectos con metodologías ágiles [33].

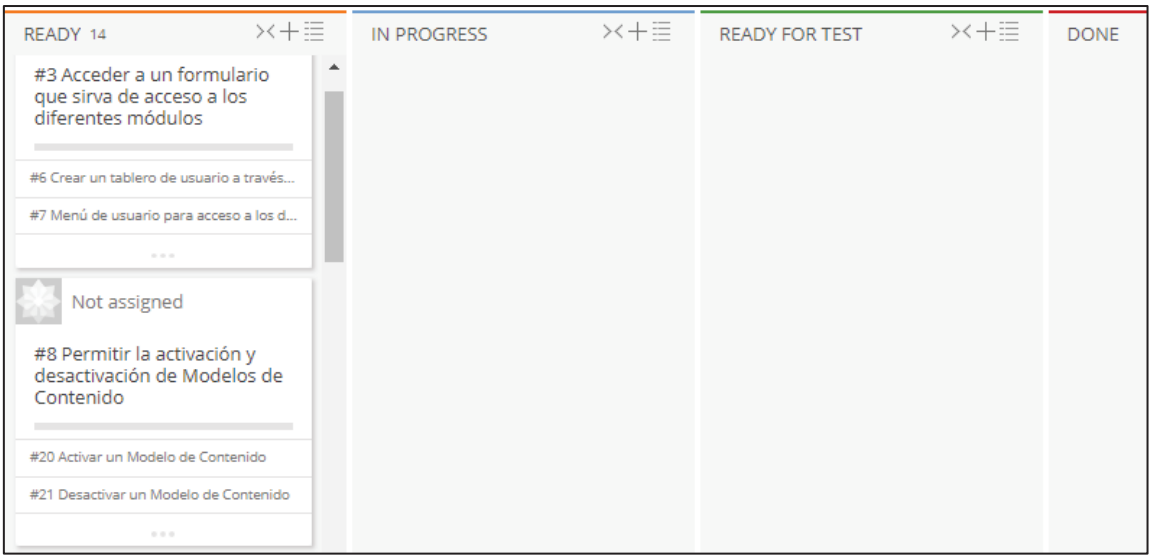


Figura 2.9 Fragmento del tablero Scrum

Tabla 2.5 *Sprint backlog* y Relación entre los *sprints* y los requerimientos

Sprint	Nombre	Requerimiento
Sprint 1	Acceso a la aplicación	1. Iniciar sesión con las credenciales de Alfresco
		2. Cerrar sesión
		3. Acceder a un formulario que sirva de acceso a los diferentes módulos
Sprint 2	Visualización de contenidos de usuario (Repositorio)	4. Permitir la visualización de archivos y carpetas de usuario
		5. Permitir cargar masivamente archivos y carpetas a través de arrastrar y soltar
		6. Permitir leer y editar metadatos de archivos y carpetas
Sprint 3	CRUD[43] de Modelos de Contenido, Aspectos y Tipos	7. Permitir la creación, lectura, edición y eliminación de Modelos de Contenido
		8. Permitir la activación y desactivación de Modelos de Contenido
		9. Permitir la creación, lectura, edición y eliminación de Tipos
		10. Permitir la creación, lectura, edición y eliminación de Aspectos
Sprint 4	CRUD de Propiedades	11. Permitir la creación, lectura, edición y eliminación de Propiedades
Sprint 5	Sincronización	12. Agregar y almacenar de sub-repositorios de Alfresco en la PC
		13. Permitir la sincronización de los contenidos de Alfresco con la PC
Sprint 6	Búsqueda de contenidos basado en metadatos	14. Permitir la búsqueda de contenidos basada en tipos, aspectos, propiedades y valores de las propiedades

[43] CRUD: Crear, leer, actualizar y eliminar.

2.1.7 Diagrama de dominios

Finalmente, y para entender la relación del usuario con Alfresco se realiza un diagrama de dominios. El diagrama de dominios se lo muestra en la Figura 2.10 y permite apreciar que el principal ente, según el número de relaciones, es un Nodo (representación de un archivo o carpeta). También se muestra que las relaciones entre cada ente se relacionan al estilo UML mediante unos y asteriscos, por ejemplo, la relación entre Nodos y Tipos es de 1 a muchos, lo que significa que un Nodo es solo de un Tipo, mientras que un Tipo puede ser aplicado a muchos Nodos.

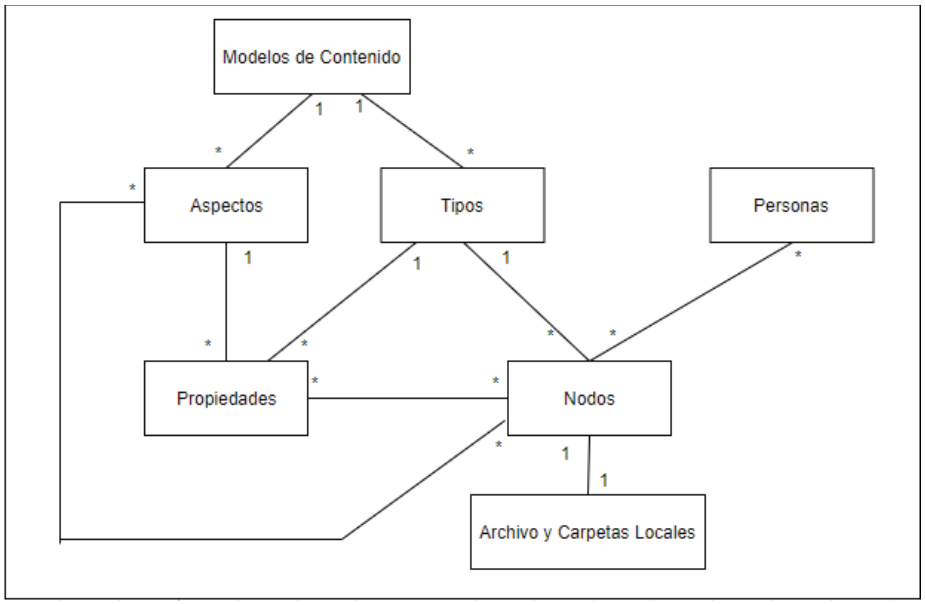


Figura 2.10 Diagrama de dominios de Alfresco

2.2 Ambiente de desarrollo

A continuación, se describen las distintas herramientas que se usan, que se han instalado y todos los pasos previos necesarios antes de escribir el código del prototipo.

Visual Studio Enterprise 2017: El lenguaje de programación que se usará es C# y la interfaz de programación visual será Windows Forms, el entorno de desarrollo integrado Visual Studio Enterprise 2017 cuenta con soporte para ambas opciones y además al momento de su instalación ofrece soporte automático de instalación del *framework* de .NET que sirve como soporte de compilación para aplicaciones basadas en C#. La descarga de este *software* se la puede realizar directamente desde su página oficial [34].

Servidor Alfresco: Antes de empezar a desarrollar, primero se instaló un servidor local del Alfresco. Para ello, se registró y descargó el *software* (versión *community*) de la página

oficial de Alfresco. Se siguió las recomendaciones de su manual de instalación [35]; los parámetros de configuración del se servidor especifican en la Tabla 2.6.

Una vez que la instalación fue completada correctamente se iniciaron los servicios `alfrescoPostgreSQL` y `alfrescoTomcat` a través de la herramienta de administración de servicios (ver Figura 2.11), desde la cual es posible gestionar los dos servicios con los cuales funciona Alfresco.

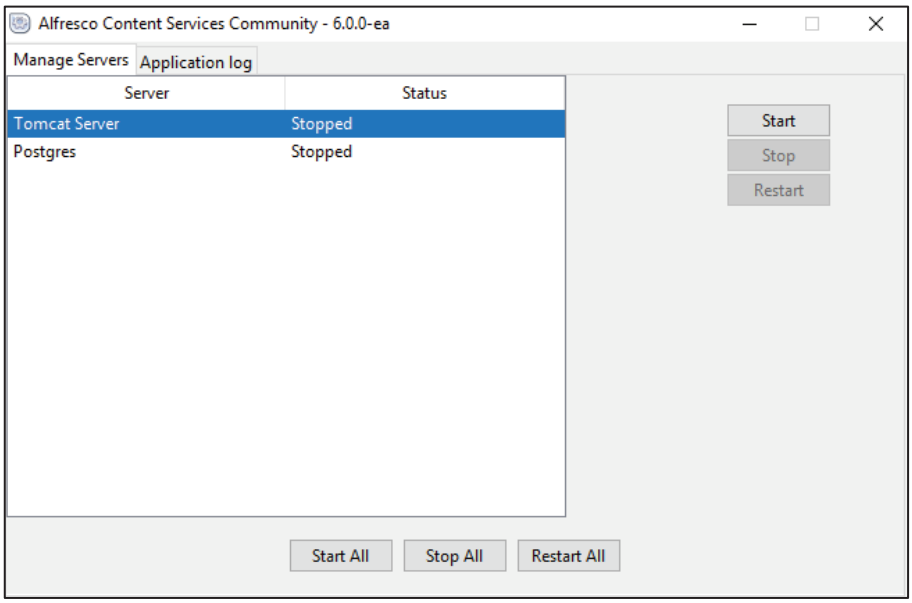


Figura 2.11 Gestor de servicios de Alfresco

Para comprobar que el servidor esté activo y funcionando de manera correcta, una vez que se ha iniciado ambos servicios de Alfresco (`alfrescoPostgreSQL` y `alfrescoTomcat`) el siguiente paso es acceder a la página web por defecto de Alfresco a través de la URL `http://localhost:8090/share`, ahí se encontrará a Alfresco Share y se podrá empezar a utilizar Alfresco.

Tabla 2.6 Parámetros de configuración del servidor Alfresco

Parámetro	**Valor**
Ubicación del servidor:	`C:\alfresco-community`
Dominio del servidor web	`127.0.0.1`
Puerto para el servidor Tomcat[44]	`8090`
Puerto para base de datos PostgreSQL[45]	`5432`
Usuario administrador	`admin`
Contraseña de administrador	`78911234`

[44] Apache Tomcat: Servidor web HTTP basado en Java.
[45] PostreSQL: Sistema de Gestión de bases de datos relacionales.

Exploración de las API RESTful de Alfresco: Una vez el servidor de Alfresco estuvo completamente funcional se empezaron a realizar interacciones sobre la ContentCMN API de Alfresco. Se eligió esta API debido a que soporta Javascript y resulta ligero ejecutar *scripts* desde la consola de usuario a través de Node.js. Se apuntó a algunos *end-points* y se realizó un análisis de las acciones que soporta la ContentCMN API. El Código 2.1 muestra un fragmento del código utilizado para probar la ContentCMN API, ahí se muestra la autenticación en Alfresco y posteriormente la inserción de un archivo sobre una carpeta especifica. Asimismo, se probó y analizó todos los *end-points* necesarios. A continuación, se detalla cada línea de código:

- Línea 1.- Se referencia la dependencia `alfresco-js-api` y se la coloca sobre la variable `AlfrescoApi`.
- Línea 4.- Se inicializa un objeto `AlfrescoApi` y se especifica la URL de conexión.
- Línea 7.- A través de la promesa[46] `login` se especifica las credenciales de autententicación: `admin` como usuario y `7891124` como contraseña. Se hace una llamada asíncrona. Dependiendo de la respuesta se mostrará un mensaje de con el *ticket* de autenticación o con un mensaje de error.
- Línea 14.- Se referencia la dependencia `fs`, utilizada para manejo de archivos.
- Línea 15.- Se inicializa un flujo de lectura apuntando al archivo `documentoTest.txt`.
- Línea 20.- Se inicializa la variable `parentFolder` que contiene el identificador del nodo en donde se insertará el archivo.
- Línea 23.- A través de la promesa `upload.uploadFile`, en la cual se especifica como parámetro de entrada el flujo de lectura inicializado en la línea 15, se realiza una llamada asíncrona insertando el archivo `documentoTest.txt` en el nodo con identificador `2958baf0-5be9-464d-aaad-c148660652c8`. Si la ejecución de la llamada asíncrona es exitosa entonces se mostrará el mensaje "*File Uploaded*" caso contrario se mostrará un mensaje de error.

Sin embargo, esta estructura organizada de Javascript disponible para la ContentCMN API no se encontró en las demás API de Alfresco, para probar las demás API se utilizó Postman en conjunto con Wireshark.

[46] Promesa: Una promesa en Javascript representa un valor proveniente de una llamada asíncrona.

```
1  var AlfrescoApi = require('alfresco-js-api');
2
3  //Se inicializa con la URL del servidor Alfresco
4  this.alfrescoJsApi = new AlfrescoApi( {hostEcm:'http://127.0.0.1:8090'});
5
6  //Se especifican credenciales del servidor Alfresco y se autentica
7  this.alfrescoJsApi.login('admin', 'passw1234').then(function (data) {
8      console.log('API called successfully Login ticket:' + data);
9  }, function (error) {
10     console.error(error);
11 });
12
13 //Se lee un archivo y se almacena su contenido en readStream
14 var fs = require('fs');
15 var readStream = fs.createReadStream('F:\\Escritorio\documentoTest.txt');
16 readStream.name = 'documentoTest.txt'
17
18 //Se especifica el id de la carpeta donde queremos que sea insertado el archivo
19 //documentoTest.txt
20 var parentFolder = '2958baf0-5be9-464d-aaad-c148660652c8';
21
22 //Se inserta el archivo en Alfresco
23 this.alfrescoJsApi.upload.uploadFile(readStream).then(function () {
24     console.log('File Uploaded');
25 }, function (error) {
26     console.log('Error during the upload' + error);
27 });
```

Código 2.1 Insertando un archivo en Alfresco a través de un *script* basado en Node.js

Una vez comprobada toda la funcionalidad de las API de Alfresco entonces se empezó a crear solicitudes HTTP basadas en C#. Las primeras pruebas se realizaron utilizando diccionarios y serializadores/deserializadores de objetos JSON, como se muestra en el Código 2.2, en este fragmento de código se creó un Modelo de Contenido para un usuario común, utilizando un diccionario como emulador de un objeto JSON, un serializador (NewtonSoft) para serializar tal diccionario en un objeto JSON y un objeto `HttpClient` para ejecutar la solicitud `POST` HTTP. Los detalles del código se muestran a continuación:

- Línea 2.- Se inicializa un objeto `HttpClient`, este será el encargado de ejecutar la llamada asíncrona.
- Línea 3.- Se agrega la cabecera HTTP Authorization, la cual contiene un *ticket* de autenticación codificado en `base64`.
- Línea 7.- Se inicializa la URI de la ContentCMN API.
- Línea 10.- Se crea un diccionario que emula a un objeto JSON.

- Línea 21.- Se serializa el diccionario en un objeto JSON.
- Línea 22.- Se inicializa un objeto `StringContent` el cual permite añadir cabeceras de tipo de codificación y tipo de contenido.
- Línea 25.- Se ejecuta la llamada asíncrona `POST` a través del método `PostAsync` que recibe como parámetros la URI de la API y el contenido a enviarse representado por el objeto del tipo `StringContent`.

```
1  //Inicialización de cliente y agregación de del header para autenticación
2  var cliente = new HttpClient();
3  cliente.DefaultRequestHeaders.Authorization =
4   new AuthenticationHeaderValue("Basic", "YWRtaW46Nzg5MTEyMzQ=");
5
6  //URI de la API
7  string URI_CMN = "http://127.0.0.1:8090/alfresco/api/-default-/private/alfresco/versions/1/cmm";
8
9  //Emulación de un objeto JSON (Modelo Personalizado) a través de un diccionario
10 var modeloPersonalizado = new Dictionary < string,
11  string > {
12   {"status", "DRAFT"  },
13   {"namespaceUri", "http://www.mymodelonoadmin.com/model2/finance/1.0"},
14   {"namespacePrefix","prefijo_noadmin_desdecsharp"},
15   {"name","modelo_noadmin_desdecsharp"},
16   {"description",""},
17   {"author", ""}
18  };
19
20 //Serialización del diccionario a objeto JSON y formateo para la solicitud HTTP
21 var modeloPersJson = JsonConvert.SerializeObject(modeloPersonalizado);
22 var modeloPersJsonHttp = new StringContent(modeloPersJson, Encoding.UTF8, "application/json");
23
24 //Ejecucion y envío de la solicitud POST
25 await cliente.PostAsync(URI_CMN, modeloPersJsonHttp);
```

Código 2.2 Solicitud `POST` creando un Modelo de Contenido desde C#

Convenciones del código: Antes de codificar y con el objetivo de tener un código limpio, ordenado y escalable se definió una estructura de directorio de proyecto como se muestra en la Figura 2.12. En la carpeta `Interfaz de Usuario` se colocarán todos los archivos correspondientes a Formularios y Controles de Usuario; en la carpeta `Modelos` se colocarán todos los archivos concernientes a las clases descriptoras del modelo de dominio; en la carpeta `Recursos` se colocarán imágenes utilizadas en la interfaz de usuario; y en la carpeta `Servicios` se colocarán las interfaces cliente-servidor, sus implementaciones y las clases de acceso a tales servicios.

Las carpetas `Modelos` y `Servicios`, están estructuradas dependiendo a la API que pertenezcan, tal como se observa en la Figura 2.13.

Para explicar la relación que existe entre las carpetas de la Figura 2.12 y las carpetas de la Figura 2.13, se toma como ejemplo la carpeta `Modelos`, esta carpeta es común para las dos estructuras. En esta carpeta están las clases descriptoras del modelo de dominio, pero

separadas en varias subcarpetas dependiendo del API que provengan. Por ejemplo, si provienen del Core API entonces sobre la carpeta `Core` se pondrán los archivos de clase que definan el modelo de la Core API.

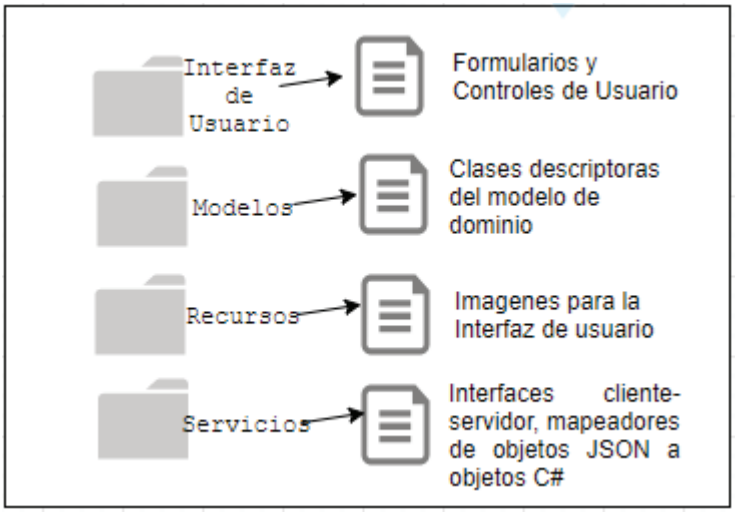


Figura 2.12 Estructura general del proyecto

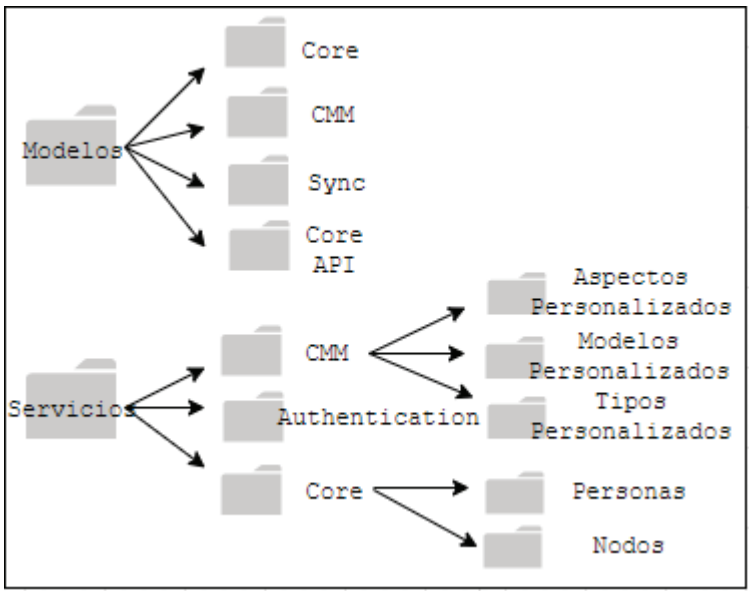


Figura 2.13 Estructura de Modelos y Servicios

Con el fin de hacer más legible el código se ha optado por utilizar ciertos estándares de nomenclatura basados en *Camel Case*[47], estos estándares se detallan en la Tabla 2.7 e indican cuando se debe aplicar determinado tipo de escritura. En la columna “Ejemplo” se muestra un ejemplo de escritura.

[47] *Camel Case*: Estándar de nomenclatura que especifica la escritura de palabras juntas (sin espacios)

Tabla 2.7 Estándares de nomenclatura utilizados en el código

Item	Tipo de escritura	Ejemplo
Variables	*Lower Camel Case*	`nodo`
Métodos	*Upper Camel Case*	`ObtenerNodo`
Formularios	Empieza con la F (mayúscula) y luego *Upper Camel Case*	`FSync`
Clases	*Upper Camel Case*	`Nodo`
Interfaces	Empieza con la I (mayúscula) y luego *Upper Camel Case*	`IPersona`
Componentes visuales	*Lower Camel Case*	`btnIngresar`

Por otra parte, cabe destacar en esta sección las características del proyecto. La Tabla 2.8 muestra las características principales del proyecto.

Tabla 2.8 Ficha del proyecto

Características del proyecto	
Nombre del ensamblado:	`TrabajoTitulacion`
Espacio de Nombres predeterminado:	`TrabajoTitulacion`
Plataforma de destino:	.NET Framework 4.6.1
Tipo de salida:	Aplicación Windows

Referencias del proyecto: Las referencias adicionales utilizadas para la realización del proyecto se descargaron a través del gestor de dependencias NuGet y se detallan en la Tabla 2.9. El Paquete `DsoFile` requirió primero instalación a nivel de sistema.

Tabla 2.9 Referencias utilizadas en el proyecto

Paquete	Nombre	Uso
`Microsoft.Rest.ClientRuntime.2.3.11`	Autorest	Estructura de proyectos basados en Swagger
`Newtonsoft.Json.11.0.2`	Newtonsoft	Serializador/Deserializador JSON
`RestSharp.106.3.1`	RestSharp	Cliente HTTP
`DocumentFormat.OpenXml.2.8.1`	DocumentFormat	Metadatos para archivos Office
`DsoFile`	DsoFile	Metadatos para archivos en general

Backup y versionamiento: Se enlazó un repositorio de GitHub al proyecto `TrabajoTitulacion`. Para ello, primero, se seleccionó la opción “Git” ubicada en el menú desplegable “Agregar al control de código fuente” ubicado en la parte inferior derecha del

Visual Studio, como se muestra en la Figura 2.14, esto permitió crear un repositorio Git local en la ubicación que sea especificada. Luego de ello se instaló la extensión de GitHub para Visual Studio y se procedió a crear un nuevo repositorio enlazado al proyecto actual, siguiendo las instrucciones disponibles en [36]. Esto permitió que cada método, clase e interfaz posean un historial de los cambios realizados. En la Figura 2.15se muestra que la interfaz `IAutenticacion` ha sido modificada por el desarrollador `jotzu3`[48] hace 4 días y que en total ha sido modificada 3 veces por un solo autor.

Figura 2.14 Agregando control al código fuente

Figura 2.15 Historial de versionamiento

2.3 Sprint Acceso a la aplicación

Este *sprint* deberá generar como entregable, la interfaz gráfica del prototipo necesaria para cumplir los requerimientos de iniciar sesión con las credenciales de Alfresco, cerrar sesión y acceder a un formulario que sirva de acceso a los diferentes módulos.

2.3.1 Interfaz gráfica

Para este *sprint* se han considerado dos *sketches* de la interfaz gráfica. El primero se refiere al formulario para el inicio de sesión de usuario y el segundo se refiere al tablero de usuario o *dashboard*. En la Figura 2.16 se muestra el *sketch* de inicio de sesión, el cual permite ingresar a la aplicación mediante un nombre de usuario y una contraseña.

La Figura 2.17 muestra el *sketch* del *dashboard*. Este *sketch* sirve como aterrizaje de la aplicación y desde aquí es posible ingresar a todas las funcionalidades de la aplicación. Cuenta con una barra de menús con cinco opciones, las mismas que están en el centro del *sketch*: 'Inicio', 'Repositorio', 'Gestor de Modelos', 'Búsqueda' y 'Sincronización'. Además, se cuenta con un último elemento de la barra de menús que es 'Usuario' el cual permite conocer la identidad de la persona que ha iniciado sesión y de hacer clic, despliega un menú que permite cerrar la sesión de usuario actual.

[48] `Jotzu3`: Cuenta GitHub del autor del Trabajo de Titulación.

La Figura 2.18 muestra la interacción entre el *sketch* de inicio de sesión y el *sketch* de *dashboard* cuando el acceso es correcto. El usuario al accionar el botón 'Ingresar' enviará sus datos de nombre de usuario y contraseña para ser validados en el servidor. En caso de que los datos sean inválidos o el servidor esté fuera de línea, el accionar del botón 'Ingresar' solo provocará un mensaje de error en rojo debajo del botón, tal y como lo muestra el *wireframe* de la Figura 2.19.

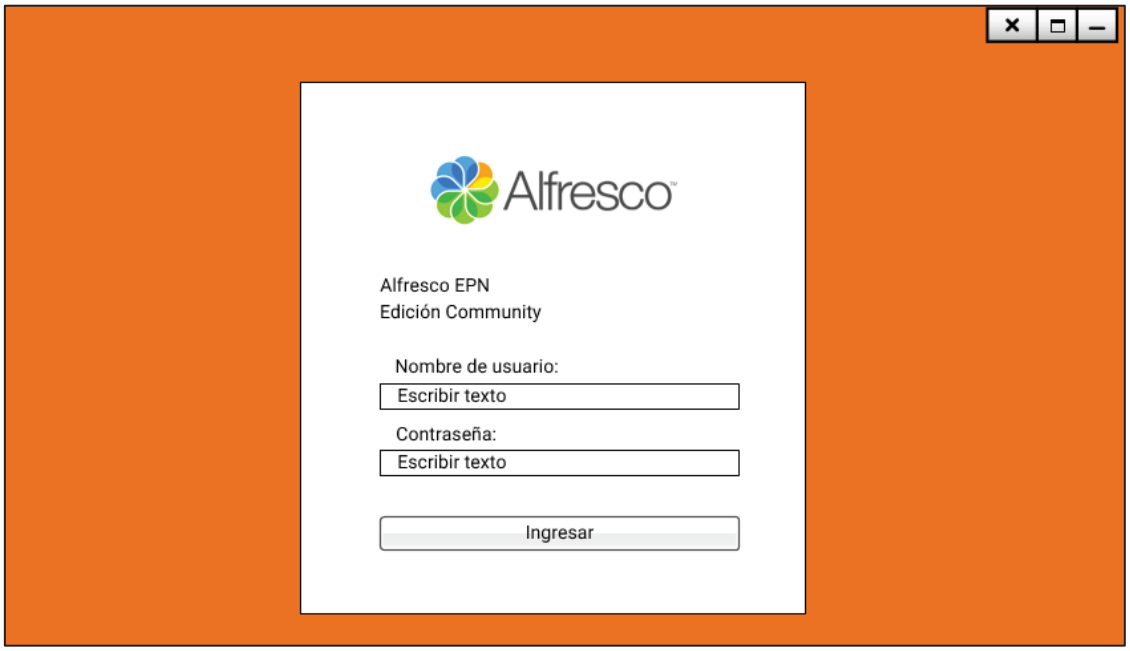


Figura 2.16 *Sketch* del *login*

Figura 2.17 *Sketch* del *dashboard*

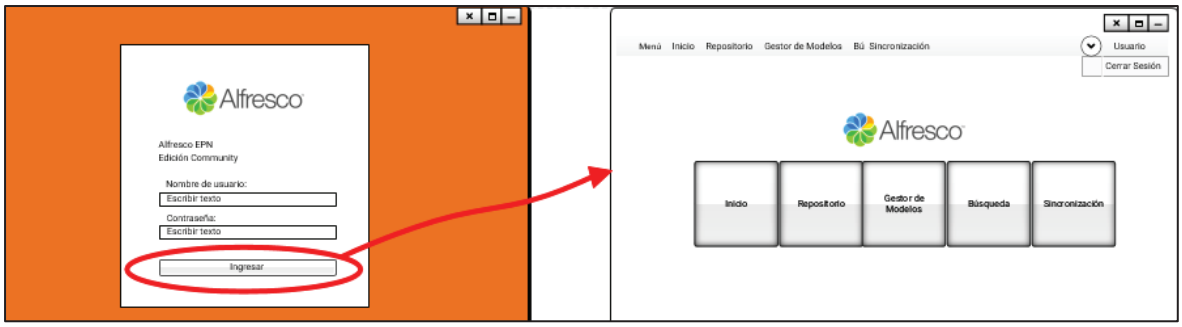


Figura 2.18 *Wireframe* de acceso correcto

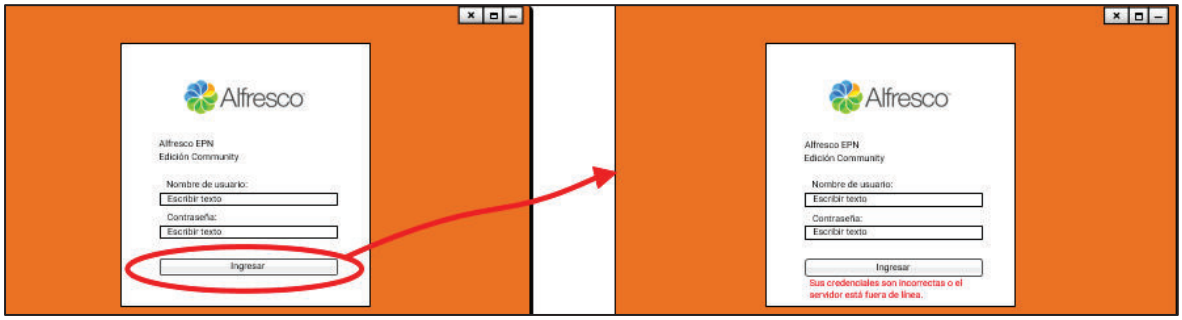


Figura 2.19 *Wireframe* de acceso erróneo

En caso de que el usuario desee cerrar su sesión, podrá terminar su sesión a través del botón desplegable 'Cerrar Sesión' de la barra de menús del *dashboard*, esta acción lo conducirá a la pantalla de Inicio de sesión, como se muestra en la Figura 2.20

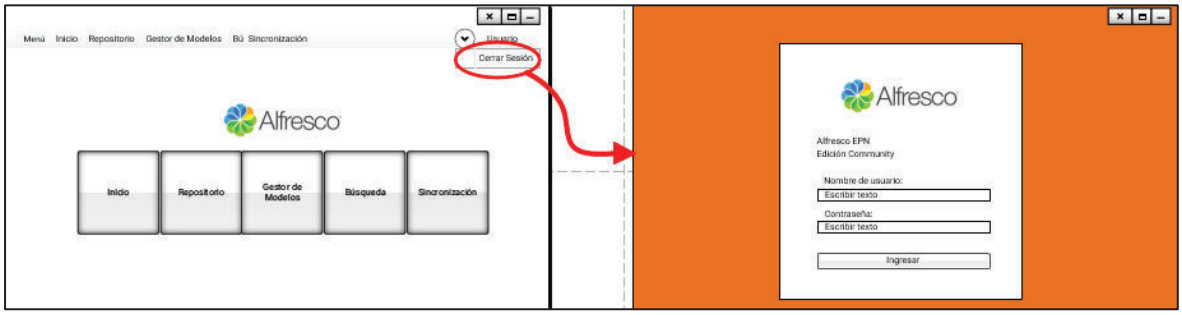


Figura 2.20 *Wireframe* cerrar sesión

2.3.2 Clases e interfaces

Para iniciar sesión y cerrar sesión se seleccionó la Authentication API de Alfresco y según sus entradas y salidas se diseñó la interfaz `IAutenticacion`. Esta interfaz surge del análisis realizado sobre la documentación de la Authentication API de Alfresco. En la Figura 2.21 se muestra en la parte superior, dos solicitudes de la Authentication API (`POST tickets` y DELETE `/tickets/userId`) con sus respectivas entradas necesarias para

que la solicitud funcione y también sus salidas. Enfocándose en el método `POST /tickets` de la Figura 2.21 se puede ver que la entrada es un objeto JSON con los atributos `userId` y `password`, mientras que la salida es un objeto JSON del tipo `entry` que contiene los atributos `id` y `userId`. Para lograr emular este comportamiento y poder enviar lo que la API específica, se crea el método C# `IniciarSesion` (parte inferior de la Figura 2.21) como parte de la interfaz de comunicación del prototipo, con los parámetros de entrada `userId` y `password`, ambos del tipo `string`, que permitirán crear el objeto JSON necesario para que la solicitud para iniciar sesión funcione de manera correcta. Como se conoce que la respuesta será un objeto JSON, es decir un `string`, entonces el retorno del método C# `IniciarSesion` será del mismo tipo, la asociación de los objetos C# y objetos JSON se detallará posteriormente.

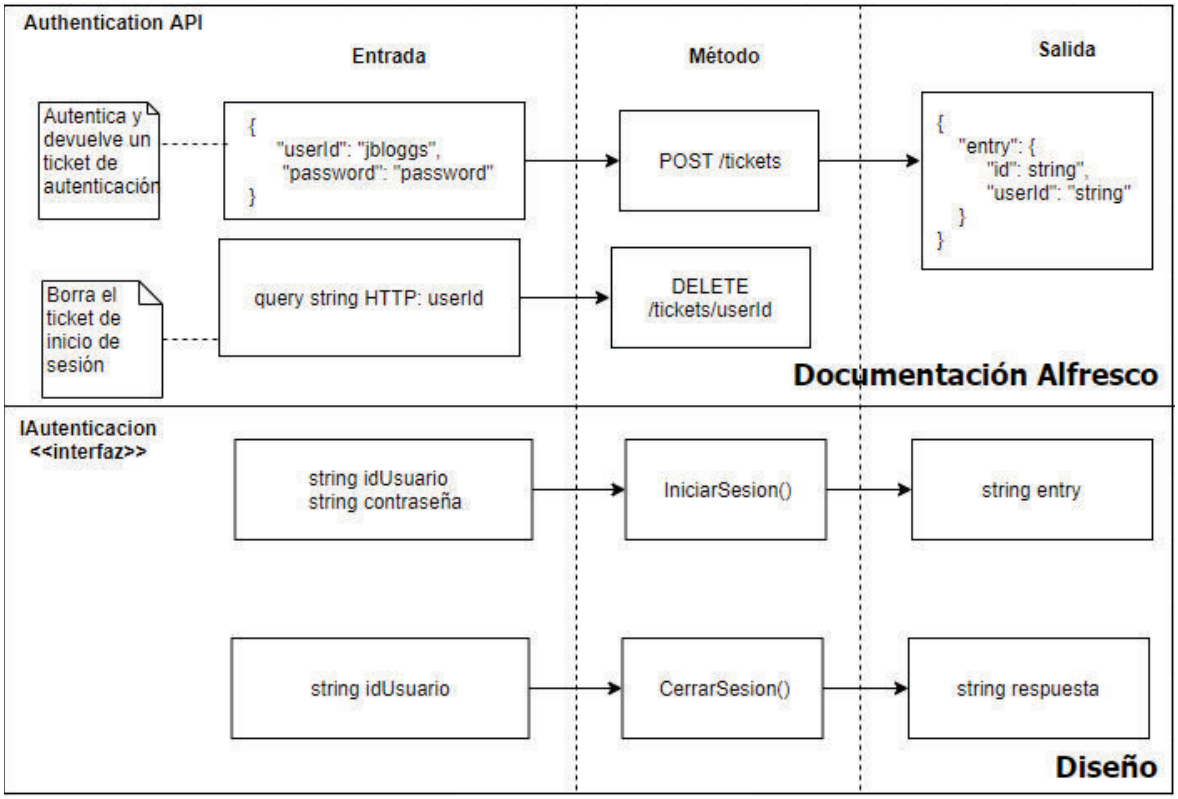


Figura 2.21 Diseño de la interfaz de comunicación para inicio y cierre de sesión

El mismo análisis se realizó para el método DELETE `/tickets/userId`, lo notorio de este método es que su entrada, `userId`, es un párametro `query string` lo que significa que viajará embebido como parte la URL de la solicitud HTTP, a pesar de ello, `userId` se traduce como un `string` para ser el parámetro de entrada del método C# `CerrarSesion`. El método HTTP no provee salida, sin embargo, de obtener una entrada errónea este método devolverá una respuesta con el código de estado y el error, es por esto que el

método `CerrarSesion` devuelve un `string`, con el objetivo de manejar algunos errores de existirlos.

Una vez diseñada la interfaz de comunicación para inicio y cierre de sesión, el siguiente paso fue diseñar la estructura de clases que respaldará a tal interfaz. La clase que implementa la interfaz `IAutenticacion` es la clase `AutenticacionServicio`. La clase encargada de asociar un objeto JSON a un objeto C# es la clase `AutenticacionStatic`, además esta clase es la única expuesta a la interfaz de usuario. `IAutenticacion`, `AutenticacionServicio`, `AutenticacionStatic` y sus respectivas relaciones se muestran en el diagrama UML de clases e interfaces de la Figura 2.22.

La clase `AutenticacionServicio` de la Figura 2.22, es la encargada de crear las solicitudes HTTP y agregar los parámetros necesarios para que estas solicitudes funcionen. Además, en esta clase se valida si la solicitud ha sido exitosa y según ello se devuelve un texto indicando el contenido de la respuesta. La clase `AutenticacionServicio` también tiene el atributo `RestClient` el cual posee todas las características para tratar con las solicitudes HTTP, además se cuenta con un atributo de nombre `URL_BASE` el cual identifica un *end-point*. Esta `URL_BASE` está formado por la dirección IP del servidor Alfresco (que está especificada en el archivo de configuración de la aplicación) y la URI de la API.

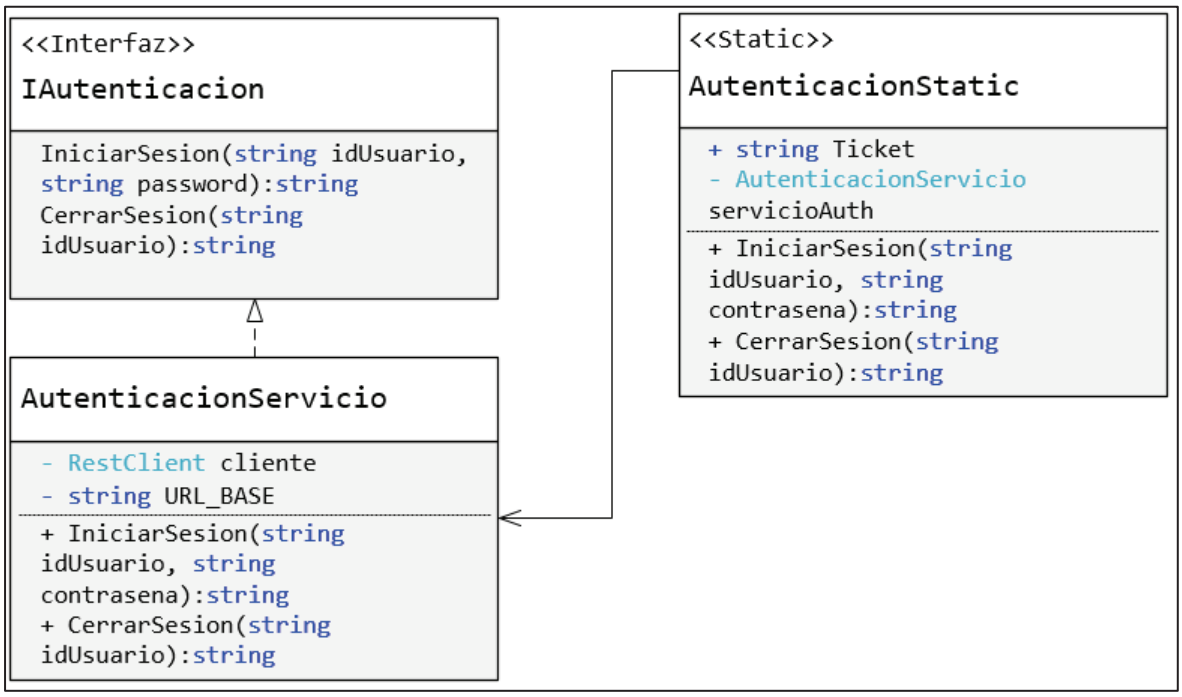


Figura 2.22 Diagrama UML de clases e interfaces para inicio y cierre de sesión

La clase `AutenticacionStatic` se presenta en la Figura 2.22. Todos sus métodos son estáticos con el objetivo de utilizar directamente un método en la interfaz sin la necesidad de instanciar un objeto. La clase `AutenticacionStatic` es la encargada de deserializar contenido JSON y asociar este contenido a objetos C#. Además, `AutenticacionStatic` contiene un mecanismo a través del cual es posible autenticar en cada solicitud posterior a un inicio de sesión, para ello se creó el atributo estático y público el cual puede ser accedido desde cualquier clase que requiera autenticarse. El *ticket* es manejado para identificar al usuario en cada solicitud, lo que permite que cada solicitud con un ticket sea reconocida en el lado del servidor como la solicitud de un usuario específico.

A parte de las acciones iniciar sesión y cerrar sesión, en este *sprint* se realiza la identificación de usuario. Para ello, en el menú de usuario se muestra el nombre de la persona que ha accedido a la aplicación. Para diseñar esta funcionalidad fue necesario utilizar el grupo *persons* de la Core API.

Basándose en la documentación de Alfresco de la Core API se diseñó la interfaz `IPersonas` (ver Figura 2.23); el método `POST /people/personId` que lee la identificación de la persona de la cadena de consulta HTTP se convierte en el método `ObtenerPersona` con el parámetro de entrada `string idUsuario`, mientras que la salida JSON de la Core API se convierte en un objeto del Tipo `entry`.

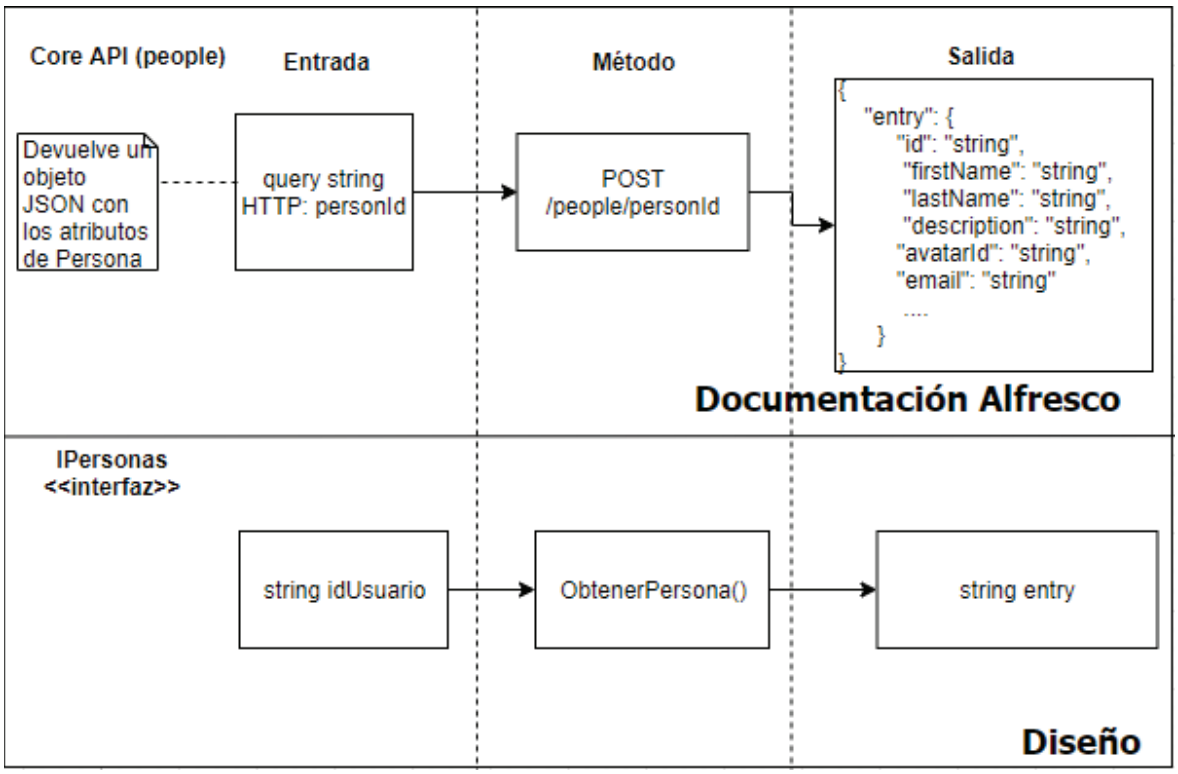


Figura 2.23 Diseño de la interfaz para identificación de usuario

La clase que implementa `IPersonas` es la clase `PersonasServicio`, la clase `PersonasStatic` es la encargada presentar los servicios de `PersonasServicio` a la interfaz de usuario, mientras que la clase `Person` es la encargada de asociar un objeto JSON `Person` a un objeto `Person` C#. La interfaz `IPersonas`, `PersonasServicio`, `PersonasStatic` y `Person` se presentan con sus respectivas relaciones y atributos, en la Figura 2.24

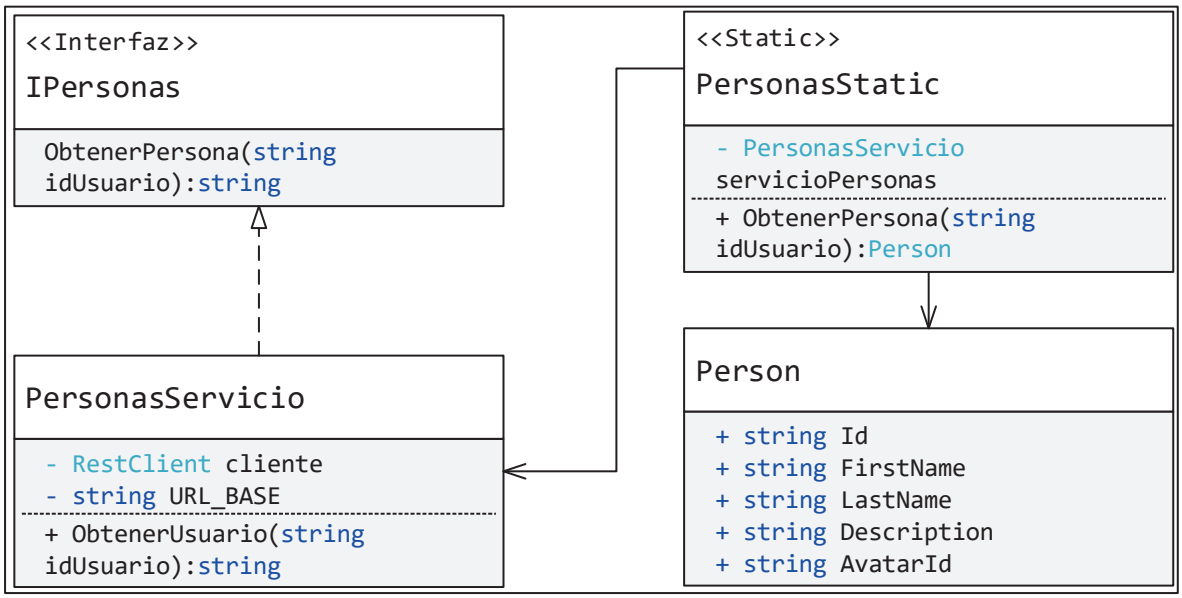


Figura 2.24 Diagrama UML de clases e interfaces para identificación de usuario

La clase `Person` surgió de la información provista por el archivo Swagger de la Core API. Una muestra de la información del archivo Swagger de la Core API se muestra en la Figura 2.25 (a). Este archivo permite generar la clase necesaria para asociar objetos JSON en objetos C#. Las propiedades (*properties*) del archivo Swagger se convierten en los atributos de la clase de C# la cual se muestra en la Figura 2.25 (b).

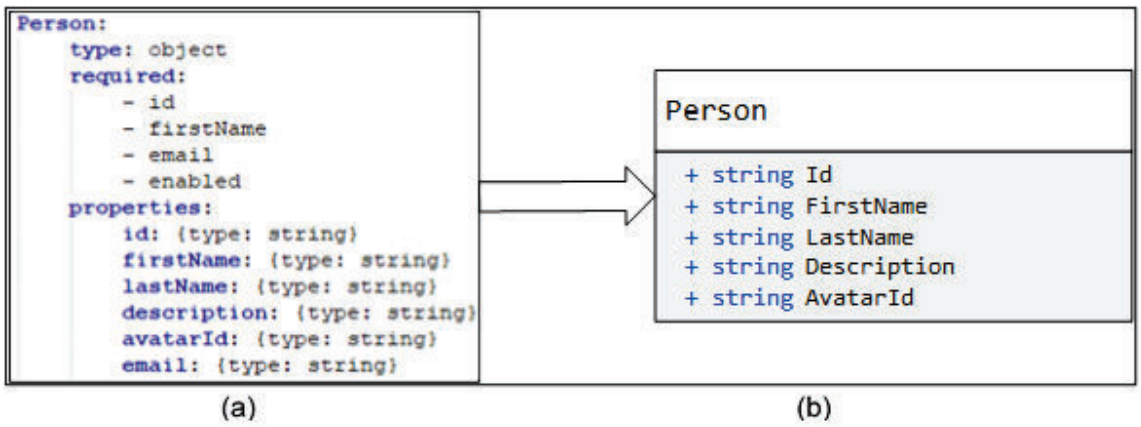


Figura 2.25 Clase `Person` en archivo Swagger asociada a clase `Person` de C#

2.3.3 Desarrollo

Para este *sprint* se comenzó codificando las interfaces y clases propuestas en la sección de Diseño. En el Código 2.3 se muestra la interfaz `IAutenticacion`, sus métodos retornan objetos del tipo `Task<string>` debido a que las llamadas que se realizarán serán asíncronas y una llamada asíncrona necesita manejar un objeto del tipo `Task<>`.

```
 9   interface IAutenticacion
10   {
11       Task<string> IniciarSesion(string idUsuario, string contrasena);
12       Task<string> CerrarSesion(string idUsuario);
13   }
```

Código 2.3 Interfaz `IAutenticación`

La clase que implementa la interfaz `IAutenticacion` es la clase `AutenticacionServicio`, un fragmento de esta clase se presenta en el Código 2.4, el método `IniciarSesion`. En la línea 30, se muestra la firma de este método, donde la palabra reservada `async` indica que este método es asíncrono y esperará en segundo plano la ejecución de una llamada `await`. Además, el Código 2.4 muestra cómo se arma una solicitud HTTP a través de una solicitud `RestRequest` provista por RestSharp; en el método `IniciarSesion` se inicializa la solicitud HTTP como `POST` (línea 32), se apunta al recurso `/tickets/` (línea 33), se especifica que el formato de la solicitud es del tipo JSON (línea 34) y en el cuerpo se envían las credenciales de usuario para autenticación como objetos JSON (línea 35), una vez que la solicitud ha sido completada entonces a través del cliente `RestClient` (que ha sido inicializado con la URL de la API) se ejecuta asíncronamente la solicitud a través del método `ExecuteTaskAsync` (línea 36); finalmente el contenido de la respuesta será verificado, y de no ser exitoso entonces se lanzará una excepción del tipo `UnauthorizedAccessException` (línea 38).

```
30   public async Task<string> IniciarSesion(string idUsuario, string contrasena)
31   {
32       var solicitud = new RestRequest(Method.POST);
33       solicitud.Resource = "tickets/";
34       solicitud.RequestFormat = DataFormat.Json;
35       solicitud.AddBody(new { userId = idUsuario, password = contrasena });
36       var respuesta = await cliente.ExecuteTaskAsync(solicitud);
37       var contenidoRespuesta = respuesta.Content;
38       if (!respuesta.IsSuccessful) throw new UnauthorizedAccessException();
39       return contenidoRespuesta;
40   }
```

Código 2.4 Clase `AutenticacionServicio`

La clase `AutenticacionStatic` es la encargada de serializar/deserializar objetos JSON en objetos C# y es la que provee métodos transparentes para la interfaz de usuario. En el Código 2.5 se muestra un fragmento de esta clase, el método estático y asíncrono `IniciarSesion`. Este método el encargado de llamar asíncronamente al método que ejecuta la solicitud HTTP (línea 23), leer el contenido de la respuesta HTTP, deserializarlo (línea 24) y tomar el `Ticket` embebido en la respuesta para asignarlo a la variable estática `Ticket` (línea 27) que servirá de autenticador global para las solicitudes HTTP posteriores. La línea 24 del Código 2.5 muestra que el tipo de objeto en el cual se deserializará la respuesta JSON de la solicitud HTTP, es del tipo `dynamic`, este tipo permite asignar el tipo de dato en tiempo de ejecución, en este caso el tipo de dato que se asignará será el tipo `entry`, que permite extraer los valores de sus atributos, basándose en su estructura JSON.

```
13  public static string Ticket { get; set; }
    0 references
14  private static AutenticacionServicio servicioAuth = new AutenticacionServicio();
15
16  /// <summary>
17  /// Autenticación de usuario
18  /// </summary>
19  /// <param name="nombreUsuario">Nombre de usuario</param>
20  /// <param name="contraseña">Contraseña de usuario</param>
    0 references
21  public async static Task<string> IniciarSesion(string nombreUsuario, string contrasena)
22  {
23      string respuestaJson = await servicioAuth.IniciarSesion(nombreUsuario, contrasena);
24      dynamic respuestaDeserializada = JsonConvert.DeserializeObject(respuestaJson);
25
26      //Se obtiene el ticket generado a partir de la respuesta
27      Ticket = respuestaDeserializada.entry.id;
28      return respuestaJson;
29      }
```

Código 2.5 Clase `AutenticacionStatic`

La interfaz gráfica para `IniciarSesión` contó con los componentes (que ejecutan una acción de usuario) mostrados en la Tabla 2.10.

Tabla 2.10 Componentes accionables del formulario para Iniciar Sesión

Nombre	Control	Acción
`btnIngresar`	`Button`	Autenticación de usuario
`txtNombreUsuario`	`TextBox`	Almacena el nombre de usuario
`txtContraseña`	`TextBox`	Almacena la contraseña de usuario

En este *sprint* también se implementó el *dashboard* de usuario, cuya funcionalidad es proveer acceso a los diferentes módulos del sistema. Para ello, se creó dos formularios, un formulario padre (`FDashboard`) del tipo `mdicontainer` que albergará a los formularios de los módulos del sistema y que además contiene un `MenuStrip` con pestañas de acceso a los módulos del sistema. El principal hijo de `FDashboard` es `FInicio` este es el formulario que funciona como tablero de usuario y a través del cual es posible acceder a todos los demás módulos del sistema.

El acceso a los diferentes módulos del sistema se realiza a través de un método por módulo, por ejemplo, en el Código 2.6 el método `AbrirInicio` (línea 67) esconde todos los formularios (línea 69) y muestra el formulario `FInicio` (línea 71). Las acciones de esconder todos los formularios y mostrar solo el formulario accedido se realiza con el fin de no volver a cargar el contenido de cada módulo y mantener su estado en caso de cambiar de módulo. Para ello se inicializó cada formulario hijo y se les asignó como padre a `FDashboard` mediante el método `AñadirFormsHijos`.

```
50  private void AñadirFormsHijos()
51  {
52      FInicio fInicio = new FInicio();
53      fInicio.MdiParent = this;
54      FRepositorio fRespositorio = new FRepositorio();
55      fRespositorio.MdiParent = this;
56      FGestorModelos fGestorModelos = new FGestorModelos();
57      fGestorModelos.MdiParent = this;
58      FSync fSync = new FSync();
59      fSync.MdiParent = this;
60      FBusqueda fBusqueda = new FBusqueda();
61      fBusqueda.MdiParent = this;
62  }
63
64  /// <summary>
65  /// Abre el formulario de FInicio
66  /// </summary>
    0 references
67  public void AbrirInicio()
68  {
69      EsconderFormsNoActuales(0);
70      MdiChildren[0].Dock = DockStyle.Fill;
71      MdiChildren[0].Show();
72  }
```

Código 2.6 Método `AñadirFormsHijos` y `AbrirInicio`

2.3.4 Entregables

Los entregables finales de este *sprint* se muestran en la Figura 2.26 (inicio de sesión) y en la Figura 2.27 (*dashboard)*.

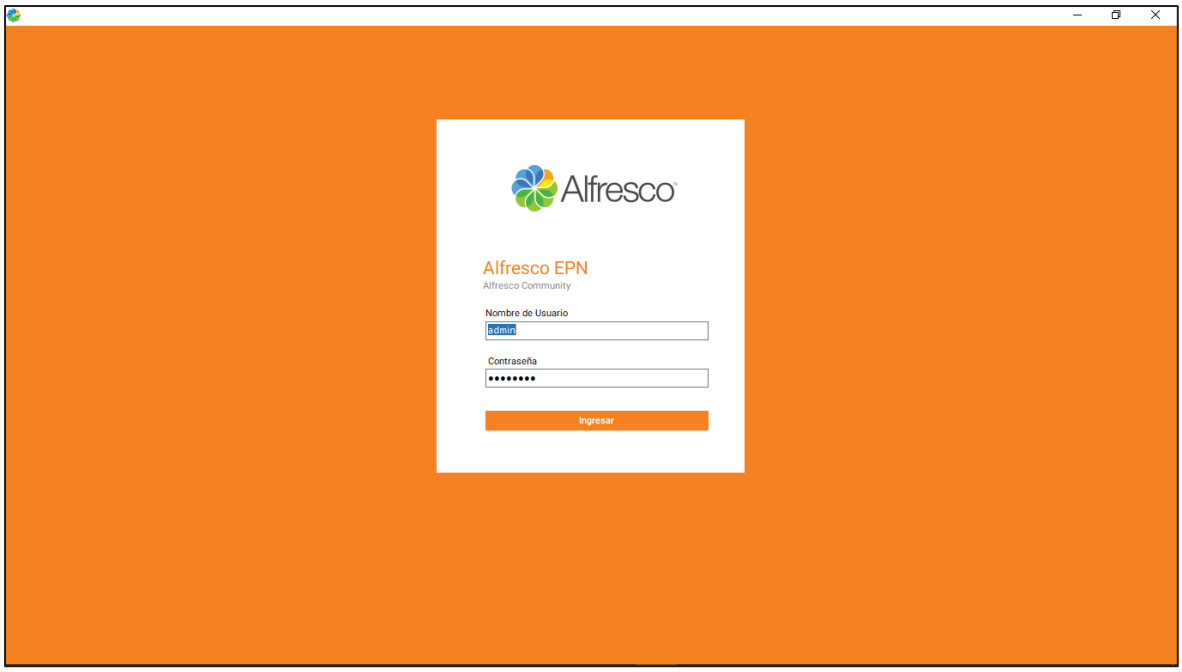


Figura 2.26 Pantalla de inicio de sesión del entregable del *sprint* 1

Figura 2.27 Pantalla del *dashboard* del entregable del *sprint* 1

2.3.5 Revisión y Retrospectiva

Este *sprint* sufrió de una revisión y retrospectiva. Se añadió las siguientes acciones:

1. Cambiar la estructura de los métodos que generan solicitudes HTTP de síncronos a asíncronos, debido a que algunas llamadas son demoradas y se necesita contar

con un mecanismo no bloqueante que ejecute las llamadas demoradas en segundo plano.

2. Añadir un formulario de carga para mostrarse cuando las llamadas sean demoradas. Este formulario se lo muestra antes de una llamada asíncrona y se lo cierra después de la misma, así mientras la llamada se está completando este formulario aparecerá como una ventana de carga.

3. Manejar las excepciones por códigos de estado y no solo por `UnauthorizedException`.

4. Utilizar la propiedad `Dock` de cada control para que el prototipo se adapte a las diferentes resoluciones de las pantallas

2.4 Sprint Visualización de contenidos de usuario

En este *sprint* se tendrá como entregable, la interfaz gráfica del prototipo necesaria para cumplir los requerimientos que permiten visualizar archivos y carpetas del usuario, permitir cargar masivamente archivos y carpetas a través de arrastrar y soltar, y permitir leer y editar metadatos de archivos y carpetas

2.4.1 Interfaz gráfica

Para este *sprint* se han considerado tres *sketches*, el primero maneja la visualización de archivos y carpetas del repositorio de usuario (árbol de nodos), el segundo se encarga de desplegar las principales propiedades de un contenido (visualización del nodo seleccionado) y el tercero se encarga de la lectura, edición y adición de metadatos a archivos y carpetas.

El *sketch* que maneja la visualización de archivos y carpetas del repositorio de usuario (árbol de nodos) se muestra en la Figura 2.28.

Para llegar al *sketch* de la Figura 2.28 es necesario hacer clic en la barra de menús, opción Repositorio o desde el *dashboard* de usuario hacer clic sobre el botón repositorio tal como se muestra en la Figura 2.29.

El *sketch* que maneja la presentación de un contenido se muestra en la Figura 2.30, este *sketch* es del tipo Control de Usuario ya que se lo deberá replicar por cada archivo o carpeta hija del nodo seleccionado a través del control `Treeview`, tal y como se muestra en el *wireframe* de la Figura 2.31. Además, permitirá navegar dentro de la carpeta a través de

un clic sobre “Nombre”, permitirá descargar el contenido del archivo a través de un clic sobre “Descargar” y permitirá visualizar y editar metadatos a través de “Ver propiedades”.

Figura 2.28 *Sketch* para navegar entre archivos y carpetas

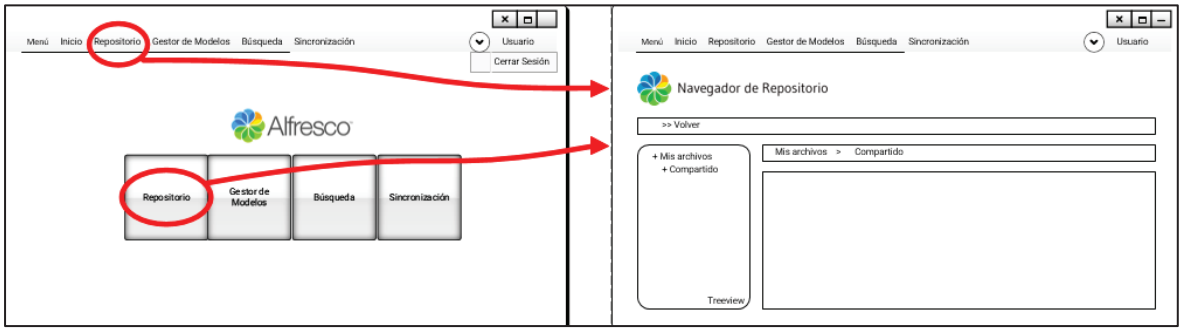


Figura 2.29 *Wireframe* de acceso al repositorio

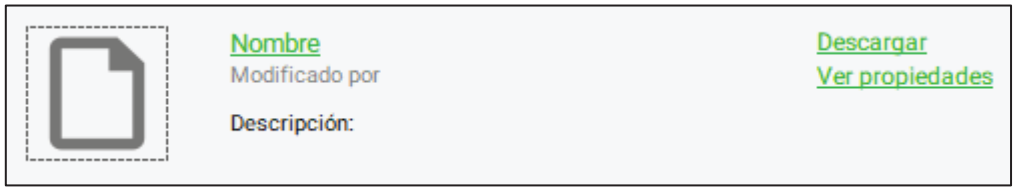


Figura 2.30 *Sketch* para la presentación de un contenido

Además, se creó un *sketch* que será del tipo cuadro de diálogo y permitirá visualizar metadatos de cada nodo y editar tales metadatos, para ello cuenta con dos botones. El primero “Editar” que sirve para activar la edición de todas las propiedades y el segundo

“Aceptar” que toma los valores de metadatos presentes en las cajas de texto y los actualiza en el servidor de Alfresco, tal y como se muestra en la Figura 2.32.

Figura 2.31 *Wireframe* de presentación de un contenido

Para poder visualizar o editar metadatos basta con hacer clic en la opción “Ver propiedades” de la Figura 2.30, tal y como se muestra en el *wireframe* de la Figura 2.33. Al realizar esta acción se mostrará la interfaz la Figura 2.32 como cuadro de diálogo.

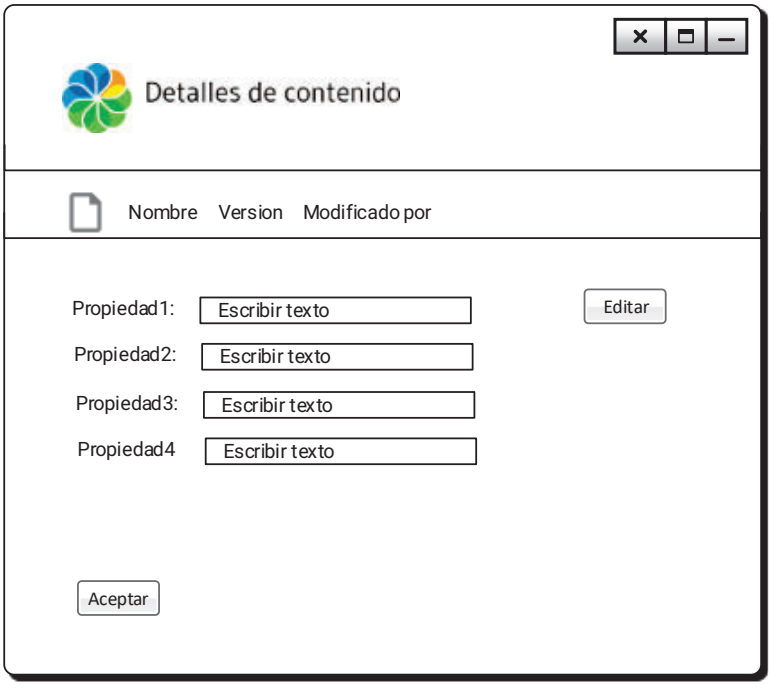


Figura 2.32 *Sketch* para visualización y edición de metadatos

Figura 2.33 *Wireframe* para visualizar y editar metadatos

2.4.2 Clases e interfaces

Para cumplir con los requisitos de este *sprint* se diseñó una interfaz de comunicación cliente-servidor. Para ello se seleccionó la API adecuada y cada *end-point* fue asociado a un método de la interfaz `INodo`. Como se muestra en la Figura 2.34, primero se analizó la documentación de Alfresco de la Core API y el método `GET /nodes/nodeId` fue asociado al método C# `ObtenerNodo` que tendrá como entrada un tipo `string` (debido a que la entrada en el método símil HTTP es de tipo `string` también) y devolverá un objeto `Nodo` (debido a que la salida en el método símil HTTP es de tipo `Node`)

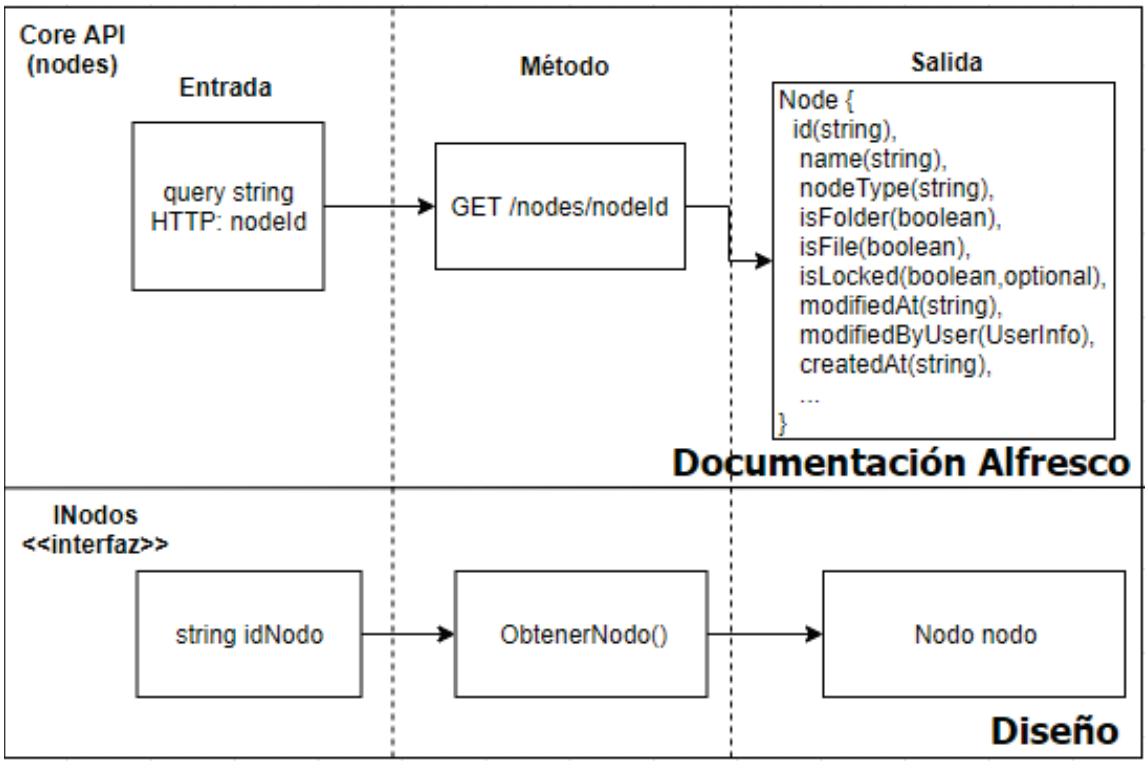


Figura 2.34 Diseño de la interfaz de comunicación para obtener un Nodo

La interfaz `INodo`, su clase implementadora (`NodoServicio`) y su clase de acceso a la interfaz gráfica (`NodoStatic`) se muestran en la Figura 2.35. La interfaz `INodo` es la encargada de proveer las definiciones de los métodos que se crearán. La clase `NodoServicio` es la clase que implementa la interfaz y sus métodos son los encargados de encapsular la creación de las diferentes solicitudes HTTP. La clase `NodoStatic` es la encargada de mapear objetos JSON a objetos C#, por ello está asociada a la clase `Nodo`. La clase `Nodo` pertenece al modelo de dominios de la aplicación y contiene la definición JSON de un nodo.

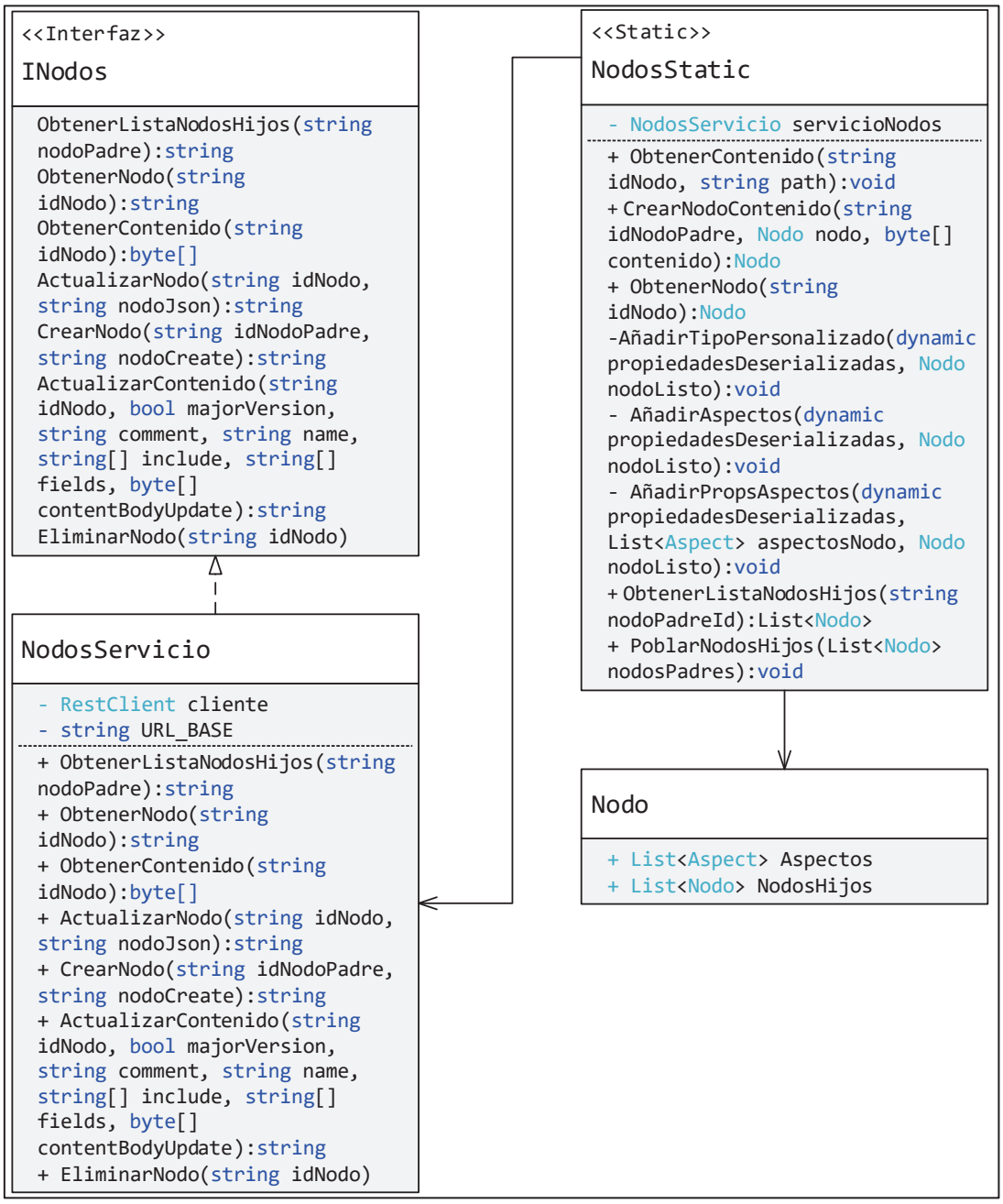


Figura 2.35 Diagrama UML para visualización de contenidos de Usuario

Para asociar objetos JSON a objetos C# fue necesario crear un diseño estructural de clases (Figura 2.36), en este diseño estructural, la clase `Nodo` contiene los atributos de un archivo o carpeta y a través de su atributo `properties`, es capaz de almacenar metadatos para ese archivo o carpeta. Además, los atributos `Aspectos` y `NodosHijos` de la clase `Nodo` han sido añadidos para brindar un acceso a través de objetos a los Aspectos y a los nodos hijos de cada nodo.

Además, se diseñó la clase `Aspect`, la cual es la encargada de asociar los aspectos de un Nodo (Lista de `string`) a una lista de objetos `Aspect`. La clase `Aspect` está formada por varios atributos, uno de ellos es el atributo `Properties` el cual contiene las propiedades (Lista de `Property`) pertenecientes a un aspecto. La clase `Property` permite tener una estructura para cada metadato de un archivo o carpeta, es decir el nombre de la propiedad, su identificador único o su valor.

Adicionalmente, se diseñó un algoritmo para recursivamente obtener el repositorio. Tal algoritmo se muestra en el diagrama de actividades de la Figura 2.37.

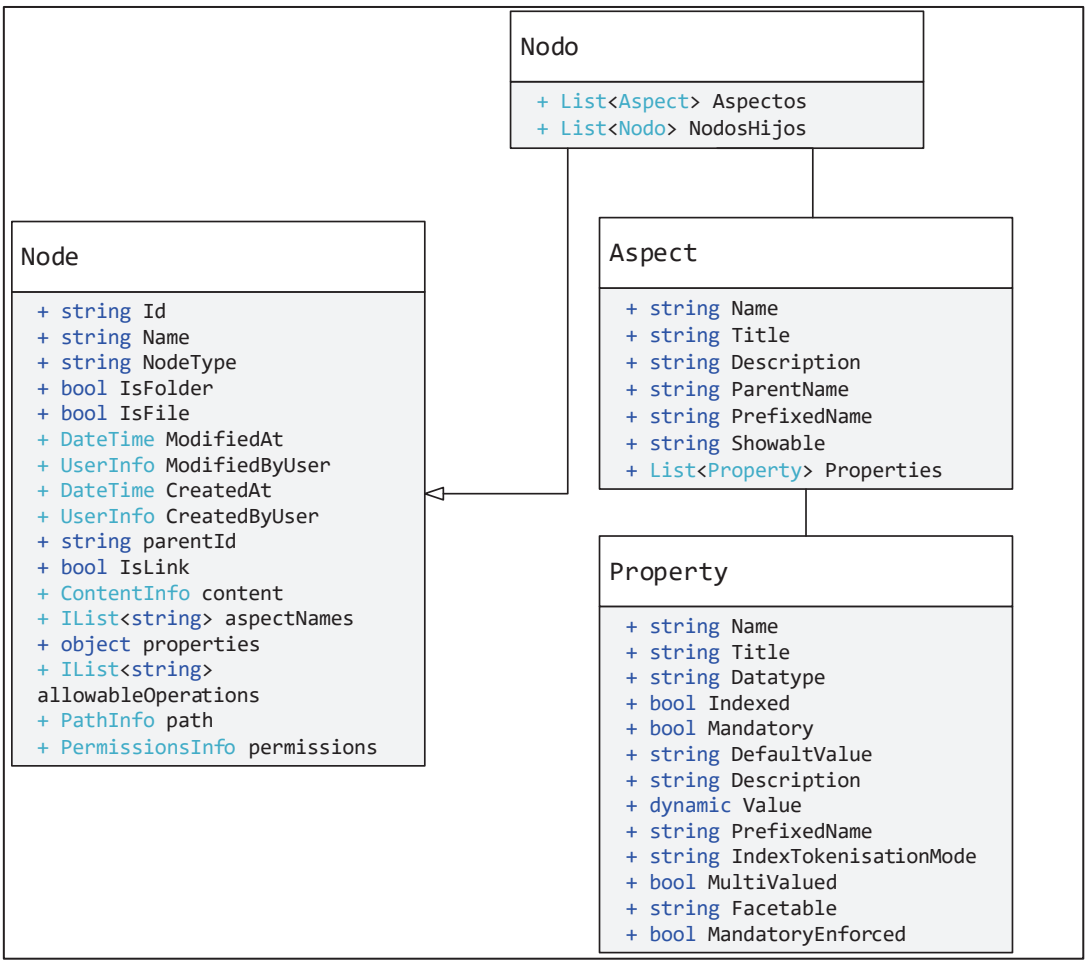


Figura 2.36 Diagrama UML de clases para manejar un Nodo

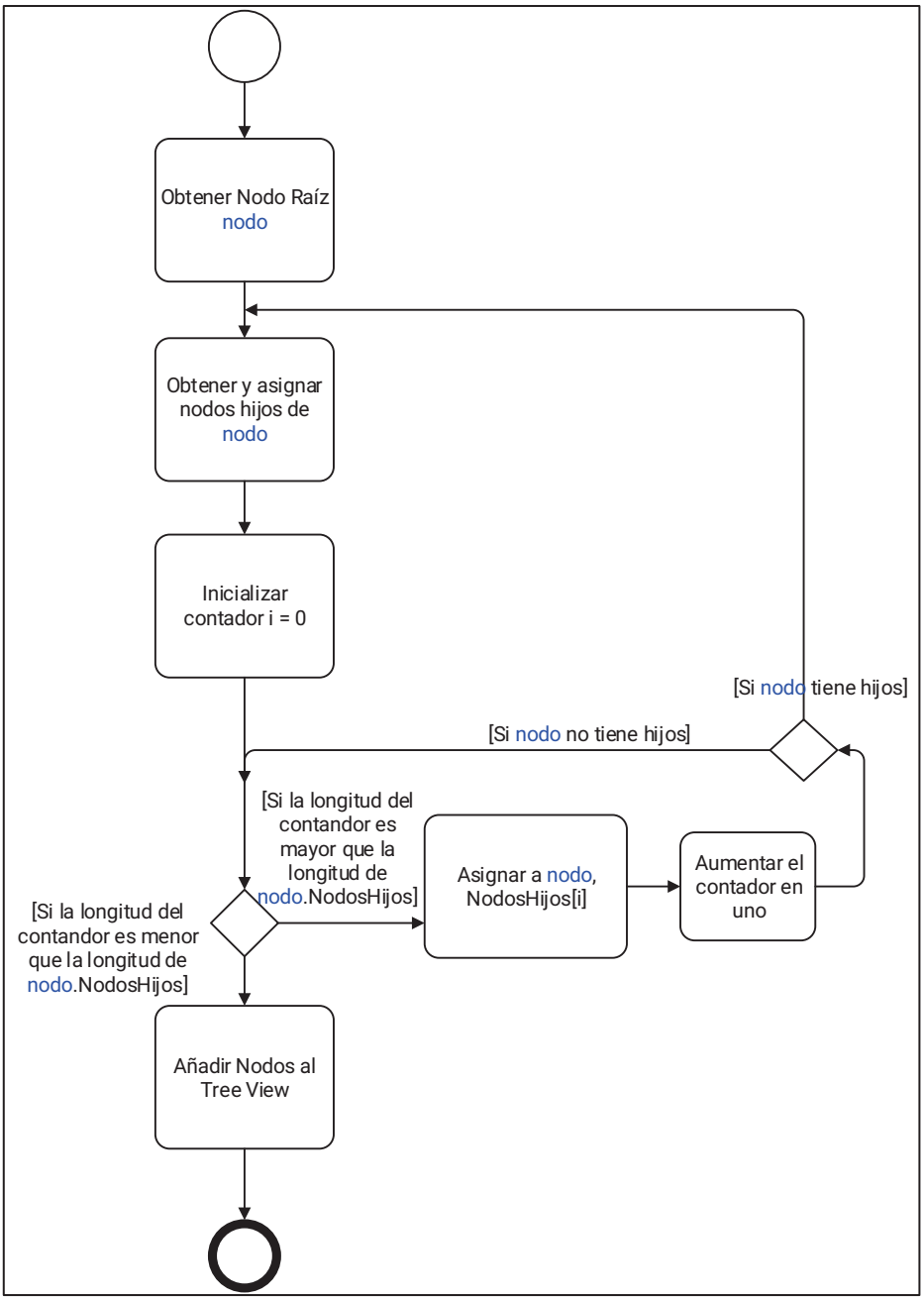


Figura 2.37 Diagrama de Actividades para obtener recursivamente el repositorio

2.4.3 Desarrollo

En este *sprint* se codificó la interfaz `INodo` (Código 2.7) según el diseño del diagrama de la Figura 2.35. Esta interfaz maneja únicamente tipos `string` y `byte`. Los tipos `string`, aunque no se especifica, deben ser del tipo JSON. Las operaciones que se manejarán en la implementación de esta interfaz deberán ser asíncronas por ello el tipo de retorno de cada método de la interfaz `INodo` debe devolver un objeto del tipo `Task<>` con el objetivo de manejar llamadas en segundo plano, posteriormente.

```
5   interface INodos
6   {
7       Task<string> ObtenerListaNodosHijos(string nodoPadre);
8       Task<string> ObtenerNodo(string idNodo);
9       Task<byte[]> ObtenerContenido(string idNodo);
10      Task<string> ActualizarNodo(string idNodo, string nodoJson);
11      Task<string> CrearNodo(string idNodoPadre, string nodoCreate);
12      Task<string> ActualizarContenido(string idNodo, bool majorVersion,
13          string comment, string name, string[] include, string[] fields, byte[] contentBodyUpdate);
14      Task<string> EliminarNodo(string idNodo);
15  }
```

Código 2.7 Interfaz `INodos`

El Código 2.8 muestra un fragmento de la clase `NodosServicio` la cual implementa de la interfaz `INodos`, en específico, se muestra el método `CrearNodo` en el cual se inicializa una solicitud `POST RestRequest` (línea 74) a la URI `nodes/idNodoPadre/children` (línea 75), la cual envía el Nodo a crearse como tipo `application/json` en el cuerpo de la solicitud (línea 76) y para autenticación se añade un ticket de autenticación como parámetro `query string` (línea 77). La línea 78 muestra la ejecución de la llamada asícrona HTTP y en la línea 79 se toma el cuerpo de la respuesta para almacenarlo en la variable `contenidoRespuesta` y devolverlo de ser exitoso.

```
66  /// <summary>
67  /// Crea un nodo
68  /// </summary>
69  /// <param name="idNodoPadre">Id del nodo padre</param>
70  /// <param name="nodoCreate">nodo JSON a crear</param>
71  /// <returns></returns>
72  public async Task<string> CrearNodo(string idNodoPadre, string nodoCreate)
73  {
74      var solicitud = new RestRequest(Method.POST);
75      solicitud.Resource = "nodes/" + idNodoPadre + "/children";
76      solicitud.AddParameter("application/json", nodoCreate, ParameterType.RequestBody);
77      solicitud.AddQueryParameter("alf_ticket", AutenticacionStatic.Ticket);
78      IRestResponse respuesta = await cliente.ExecuteTaskAsync(solicitud);
79      var contenidoRespuesta = respuesta.Content;
80      if (!respuesta.IsSuccessful) throw new UnauthorizedAccessException();
81      return contenidoRespuesta;
82  }
```

Código 2.8 Fragmento de la clase `NodosServicio`, método `CrearNodo`

Para exponer la interfaz de comunicación a la interfaz de usuario, se utilizó la clase estática `NodosStatic`, esta clase se encarga de asociar objetos y de serializar y deserializar el contenido JSON de tal manera que la codificación en la interfaz de usuario se maneje directamente con objetos C#. En el Código 2.9 se muestra el método `CrearNodo`, en este método se crea un objeto de la clase `NodeBodyCreate` a partir del nodo recibido (líneas

337-342), se utiliza esta clase porque es la aceptada por la API. Luego se serializa este objeto para obtener un `string` JSON (línea 343) y se utiliza el objeto `servicioNodos` de la clase `NodoServicio` para ejecutar la solicitud HTTP (línea 344). Por último, se deserializa el contenido recibido (de `Node` en formato JSON a `Nodo` en formato C#) a través del método `DeserializarNodoJson` y se retorna el objeto de la clase `Nodo` (línea 345).

```
329   /// <summary>
330   /// Crea un nodo en el servidor de Alfresco
331   /// </summary>
332   /// <param name="idNodoPadre">Id del nodo padre (carpeta)</param>
333   /// <param name="nodo">Objeto nodo de la clase Nodo a crearse</param>
334   /// <returns></returns>
      0 references
335   public async static Task<Nodo> CrearNodo(string idNodoPadre, Nodo nodo)
336   {
337       NodeBodyCreate nodeBodyCreate = new NodeBodyCreate
338       {
339           Name = nodo.Name,
340           NodeType = nodo.NodeType,
341           Properties = (Dictionary<string, string>)nodo.Properties
342       };
343       string nodeBodyCreateJson = JsonConvert.SerializeObject(nodeBodyCreate);
344       string respuestaJson = await servicioNodos.CrearNodo(idNodoPadre, nodeBodyCreateJson);
345       return DeserializarNodoJson(respuestaJson);
346   }
```

Código 2.9 Fragmento de la clase estática `NodosStatic`

La interfaz de usuario para visualización de contenidos necesitó dos formularios (`FRepostiorio` y `FDetallesContenido`) y un control de usuario (`CtrluContenido`).

En el Código 2.10, se muestra la implementación, sobre el manejador del evento `Load` del formulario `FRepositorio`, en donde se implementa el algoritmo de la Figura 2.37 para recursivamente obtener el repositorio de usuario y agregarlo al control `treeview` de este formulario. Para ello primero se obtiene el nodo raíz (línea 29), se obtienen y asignan los hijos del nodo raíz (línea 30) y recursivamente se empieza a obtener los nodos con sus hijos a través del método `PoblarNodosHijos` (línea 35). En este punto mientras se realiza la carga recursiva, se presenta una pequeña ventana de carga a través del formulario `FLoading` (como los métodos son asíncronos es sencillo parar la ejecución sin bloquear la aplicación). Una vez que todos los nodos del repositorio se han cargado entonces a través del método `Add` de la lista de nodos del control `treeview` se añade el nodo raíz (línea 38-39). Para añadir los demás nodos recursivamente al control `Treeview` se utiliza el método `AñadirNodosTV` (línea 40) el cual también utiliza el método `Add` de la lista de nodos para realizar la adición recursiva.

```
21  private async void FRepositorio_Load(object sender, EventArgs e)
22  {
23      try
24      {
25          FLoading fPrincipalLoading = new FLoading();
26
27          //Se obtiene la lista de los nodos hijos de root (1er nivel) y el nodo root
28          List<Nodo> nodosDeRoot;
29          nodoRoot = await NodosStatic.ObtenerNodo("-root-");
30          nodosDeRoot = await NodosStatic.ObtenerListaNodosHijos("-root-");
31          nodoRoot.NodosHijos = nodosDeRoot;
32
33          fPrincipalLoading.Show();
34          //Se pobla recursivamente todos los nodos
35          await NodosStatic.PoblarNodosHijos(nodosDeRoot);
36
37          //Se agrega los nodos al treeview
38          treeViewRepositorio.Nodes.Add(nodoRoot.Id, "Mis archivos");
39          treeViewRepositorio.Nodes[nodoRoot.Id].Tag = nodoRoot;
40          AñadirNodosTV(nodosDeRoot, treeViewRepositorio.Nodes[nodoRoot.Id]);
41          treeViewRepositorio.Refresh();
42          fPrincipalLoading.Close();
43
44      }
45      catch (UnauthorizedAccessException)
46      {
47          MessageBox.Show("Hubo un problema cargando sus archivos.");
48      }
49  }
```

Código 2.10 Obtención del repositorio remoto y poblado del contro `Treeview`

La forma en la cual se dibujó cada contenido sobre el `flowlayoutpanel` (control encargado de mostrar el contenido de cada carpeta del repositorio) se muestra en el Código 2.11. En ese fragmento de código se puede observar que se está utilizando el control de usuario `CtrluContenido` que representa a un nodo y sus propiedades principales, además sobre el control de usuario se modifican sus etiquetas de tal manera que el contenido dibujado corresponda a un nodo en específico (líneas 155-157). También, se muestra cómo se realiza el *wiring*[49] de los eventos con sus manejadores para lograr realizar acciones como descargar el contenido (línea 160), navegar sobre el contenido (línea 161) y visualizar las propiedades del mismo (línea 162).

Uno de los manejadores de evento del Código 2.11 (`LnklblPropiedades_MouseClick`) se usa para iniciar el formulario de diálogo `FDetallesContenido`. Este formulario es el encargado de mostrar las propiedades de cada contenido, editarlas y añadir nuevas.

[49] *Wiring*: Término utilizado para referirse al enlace entre un evento y su método manejador.

```
146   /// <summary>
147   /// Dibuja un nodo (CtrluContenido) y lo ubica en el flowlayoutpanel
148   /// </summary>
149   /// <param name="nodoHijo">Nodo a dibujarse</param>
      0 references
150   private void DibujarContenido(Nodo nodoHijo)
151   {
152       CtrluContenido ctrluContenido = new CtrluContenido();
153       ctrluContenido.BackColor = System.Drawing.Color.Transparent;
154       ctrluContenido.Anchor = (AnchorStyles.Top | AnchorStyles.Right);
155       ctrluContenido.LnklblNombre.Text = nodoHijo.Name;
156       ctrluContenido.LblModificado.Text = "Modificado el " +
157           nodoHijo.ModifiedAt.ToShortDateString() + " por: " + nodoHijo.ModifiedByUser.DisplayName;
158       ctrluContenido.Tag = nodoHijo;
159       flwlypanelNodosHijos.Controls.Add(ctrluContenido);
160       ctrluContenido.LnklblDescargar.MouseClick += LnklblDescargar_MouseClick;
161       ctrluContenido.LnklblNombre.MouseClick += LnklblNombre_MouseClick;
162       ctrluContenido.LnklblPropiedades.MouseClick += LnklblPropiedades_MouseClick;
163   }
```

Código 2.11 Código para dibujar un nodo sobre el panel contenedor de contenidos

2.4.4 Entregables

Los entregables del segundo *sprint* son el visualizador de contenidos (Figura 2.38) y el visualizador de detalles de contenido (Figura 2.39).

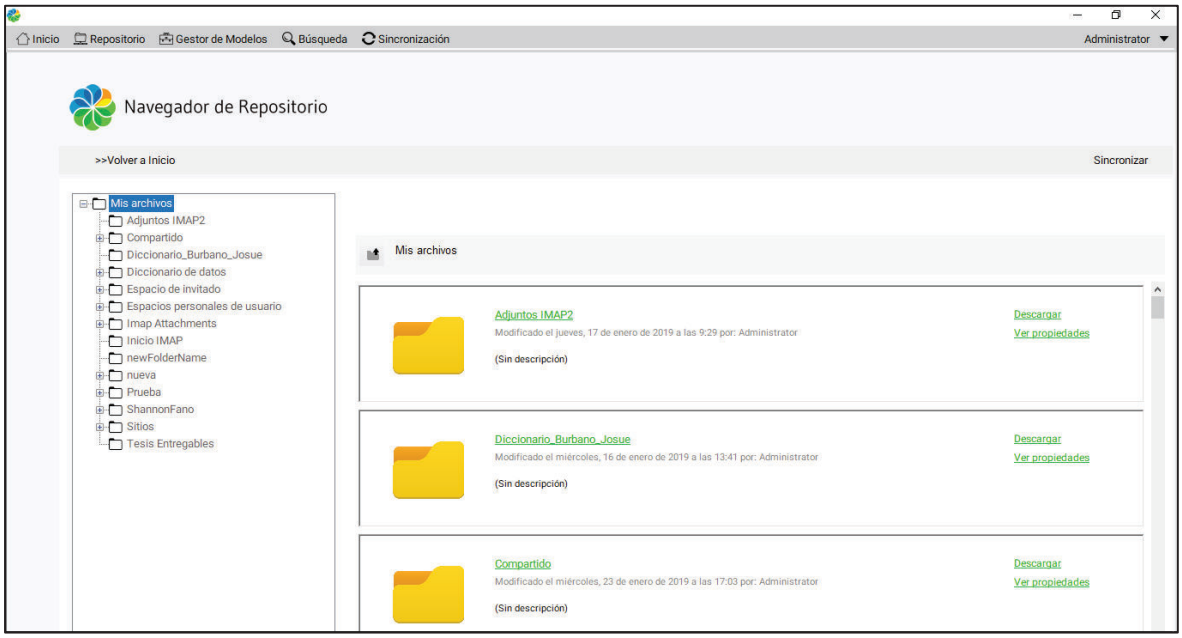


Figura 2.38 Pantalla del visualizador de contenidos del entregable del segundo *sprint*

2.4.5 Revisión y Retrospectiva

Los cambios que se realizaron a lo largo del desarrollo de este *sprint* fueron:

1. En un inicio los archivos y carpetas se mostraban en un control `treeview`, luego se decidió evitar que los archivos aparezcan en el control `treeview`. El contenido completo de cada carpeta se muestra en la parte derecha del `treeview`.

2. En un inicio las Propiedades de cada nodo se mostraban embebidas en cada archivo o carpeta, luego para poder editar las propiedades se decidió sacar las mismas a un formulario de diálogo donde sea posible editarlas y agregar nuevas.

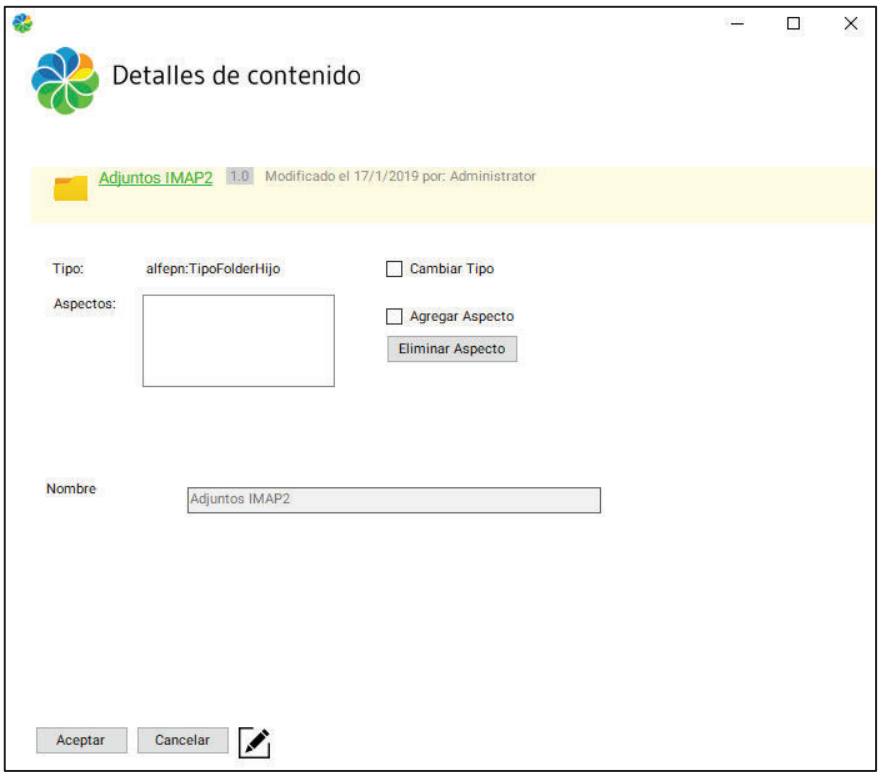


Figura 2.39 Pantalla de los detalles de contenido del entregable del *sprint* 2

2.5 Sprint CRUD de Modelos de Contenido, Aspectos y Tipos

En este *sprint* se generará como entregable, la interfaz gráfica del prototipo necesaria para cumplir los requerimientos que permitan la creación, lectura, edición y eliminación de Modelos de Contenido; permitan la activación y desactivación de Modelos de Contenido, permitir la creación, lectura, edición y eliminación de Tipos; y permitan la creación, lectura, edición y eliminación de Aspectos.

2.5.1 Interfaz gráfica

Para este *sprint* se ha considerado un formulario padre y tres formularios hijos. El *sketch* del formulario padre se muestra en la Figura 2.40 y sirve de contenedor para los demás formularios de este *sprint*.

Para llegar al formulario de la Figura 2.40 se lo puede hacer a través del *dashboard* o del menú de opciones de la aplicación, tal y como se muestra en el *wireframe* de la Figura 2.41.

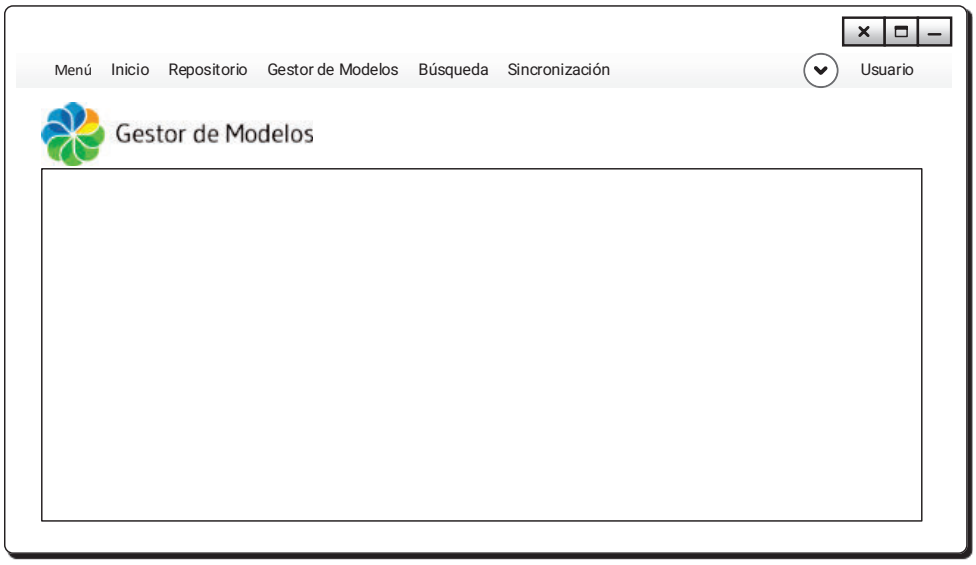


Figura 2.40 *Sketch* padre para gestionar modelos, aspectos y tipos.

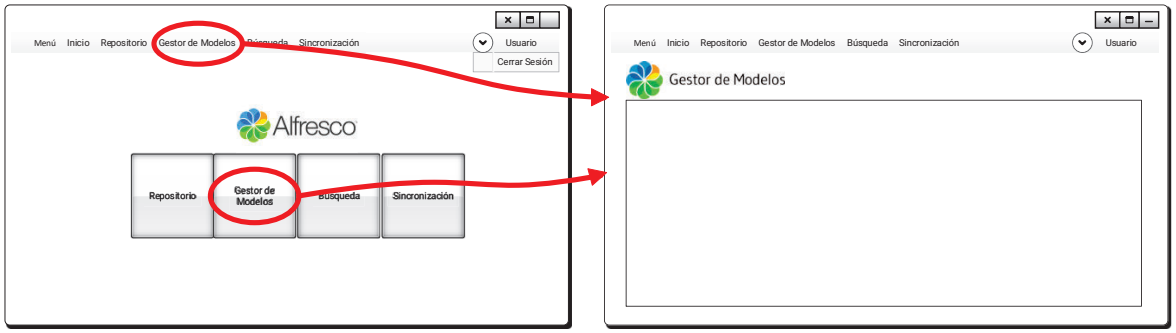


Figura 2.41 *Wireframe* para acceder al gestor de modelos

Los formularios hijos del gestor de modelos tienen una estructura similar y sirven para gestionar Modelos de Contenido (Figura 2.42), Aspectos de un Modelo de Contenido (Figura 2.43) y Tipos de un Modelo de Contenido (Figura 2.44).

En la Figura 2.42 se muestra el diseño de la interfaz de usuario para crear, actualizar, leer y eliminar Modelos de Contenido. El panel derecho “Crear Modelo” sirve para crear y editar mientras que el panel izquierdo “Modelos Actuales” sirve para leer y eliminar Modelos de Contenido de usuario y cuenta con un menú contextual que permite activar/desactivar el modelo y acceder a sus Aspectos y Tipos.

En la Figura 2.43 se muestra el *sketch* para crear, actualizar, leer y eliminar Aspectos. Al igual que en el *sketch* de la Figura 2.42, el panel derecho “Crear Aspecto” sirve para crear

y editar Aspectos, mientras que el panel izquierdo sirve para visualizar la lista de Aspectos de un Modelo de Contenido, eliminar alguno de ser necesario y acceder a las Propiedades de un Aspecto seleccionado.

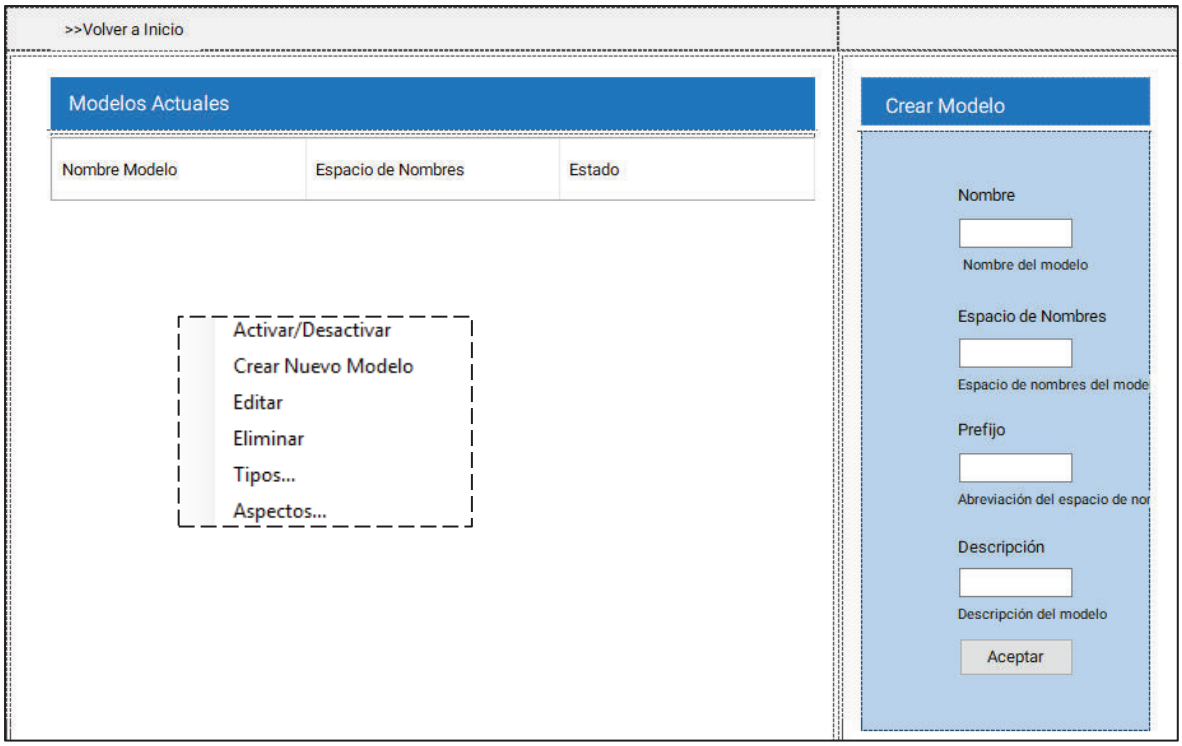


Figura 2.42 *Sketch* para gestionar Modelos de Contenido

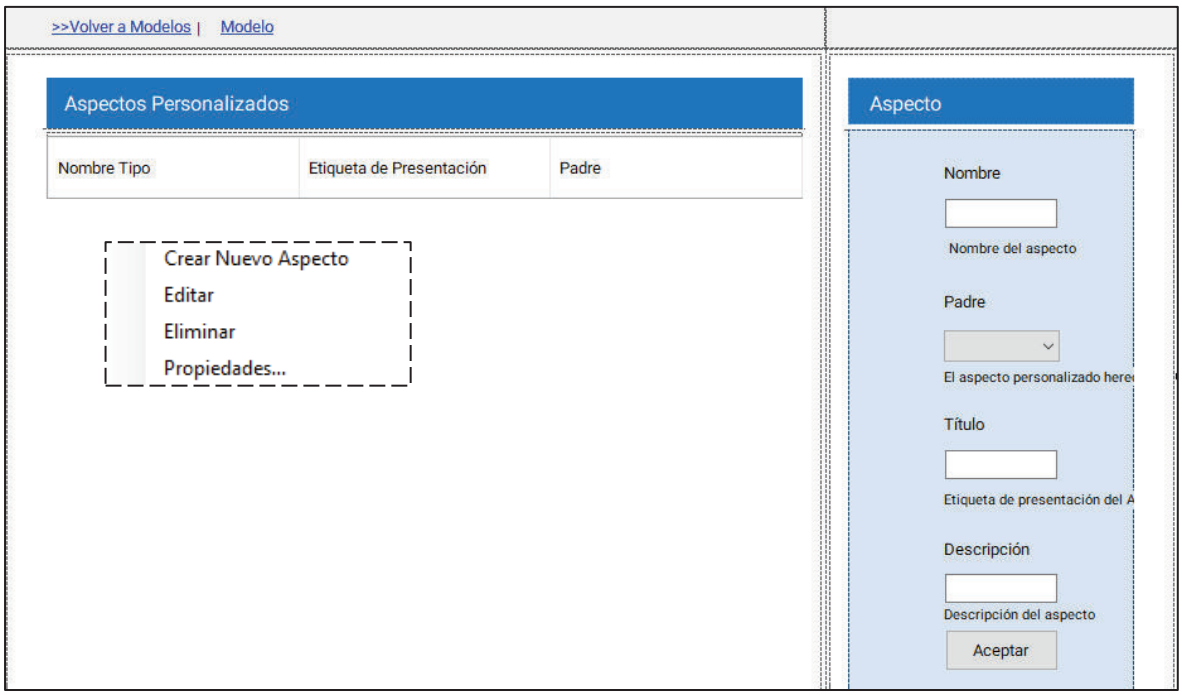


Figura 2.43 *Sketch* para gestionar Aspectos de un Modelo de Contenido

La Figura 2.44 muestra el *sketch* para crear, actualizar, leer, eliminar y acceder a los atributos de un Tipo, es decir tiene un panel derecho “Crear Tipo” desde el cual se puede crear un nuevo Tipo personalizado o se puede cargar un Tipo personalizado (seleccionado desde la lista de Tipos) para su edición. En el panel izquierdo se muestran los Tipos personalizados del usuario en una lista y este panel contiene un menú contextual a través del cual es posible eliminar o acceder a las Propiedades del Tipo. Es posible a través del botón “Volver a modelos” regresar a la pantalla donde se encuentran todos los modelos de usuario.

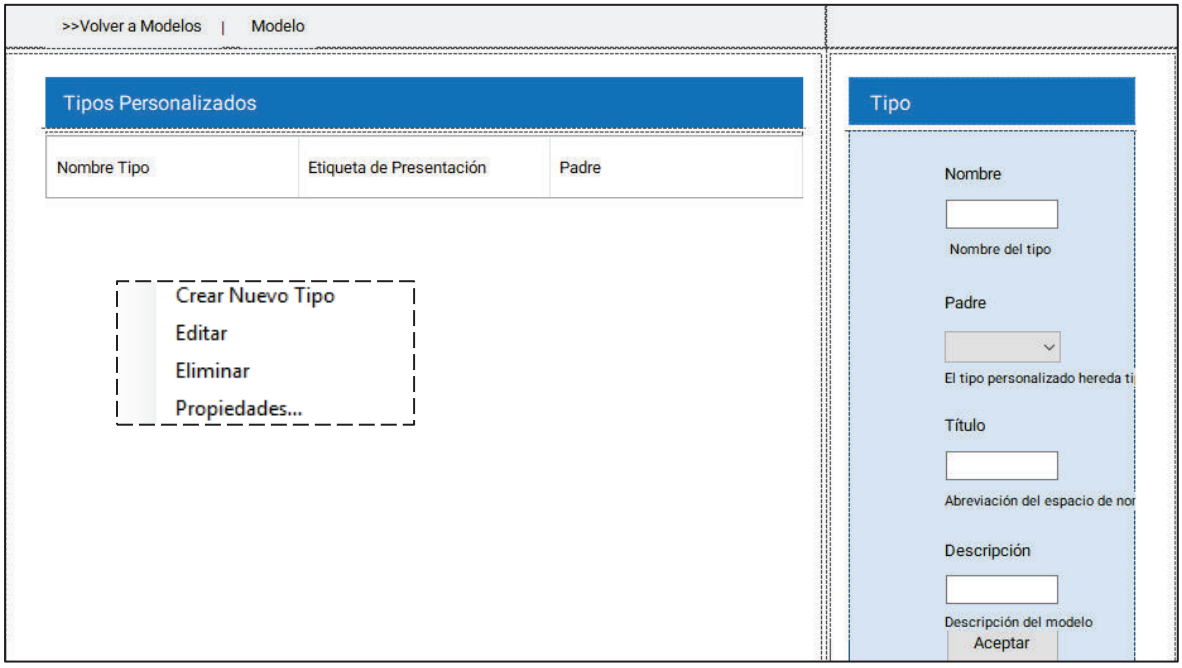


Figura 2.44 *Sketch* para gestionar Tipos personalizados de un Modelo de Contenido

La interacción entre los *sketches* de Modelos y Tipos se muestra en el *wireframe* de la Figura 2.45. Por otra parte, la interacción entre Modelos y Aspectos se muestra en el *wireframe* de la Figura 2.46.

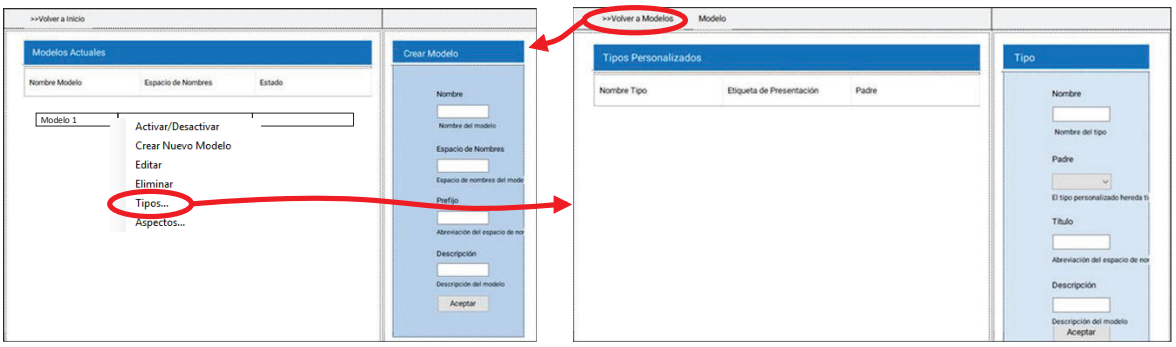


Figura 2.45 *Wireframe* entre Modelos y Tipos

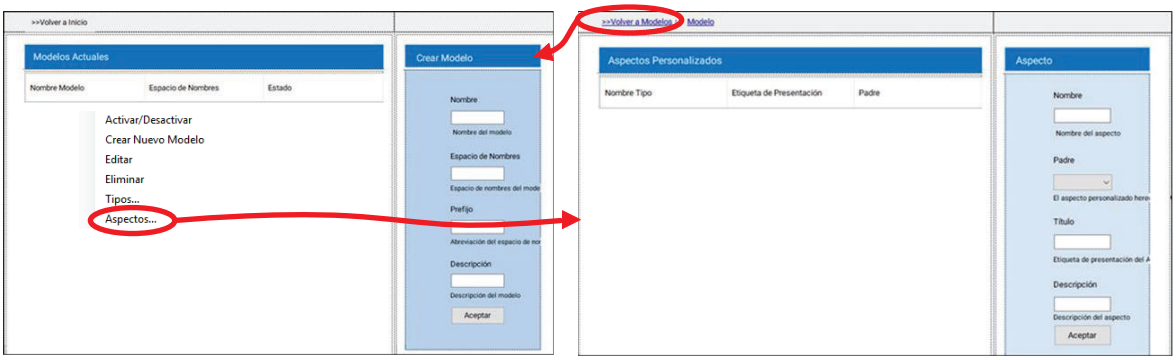


Figura 2.46 *Wireframe* entre Modelos y Aspectos

2.5.2 Clases e interfaces

Se empezó definiendo las interfaces de comunicación cliente-servidor, estas interfaces están basada en la API ContentCMN. La interfaz de comunicación para modelos se muestra en la Figura 2.48, la interfaz de comunicación para tipos en la Figura 2.49 y la interfaz de comunicación de aspectos en la Figura 2.50. Las tres interfaces conservan el principio de encapsulamiento en la implementación de la interfaz de comunicación y el acceso estático desde la interfaz gráfica, como se muestra en la Figura 2.47.

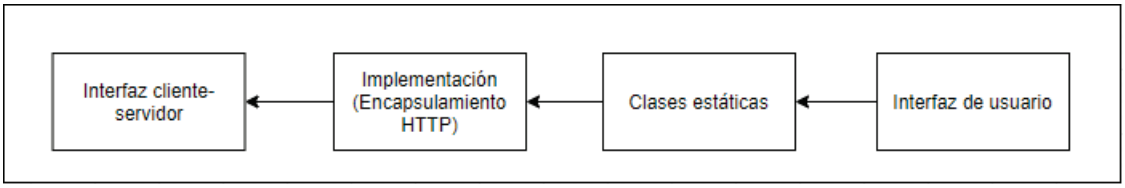


Figura 2.47 Secuencia de acceso de la interfaz de usuario a los diferentes servicios

```
<<Interfaz>>
IModelosPersonalizados
+ ObtenerModelos(): void
+ ObtenerModelo(string
nombreModelo):string
+ CrearModelo(string modelo):
string
+ ActualizarModelo(string
nombreModelo, string modelo):
string
+ ActivarModelo(string
nombreModelo):string
+ DesactivarModelo(string
nombreModelo):string
+ EliminarModelo(string
nombreModelo):string
```

Figura 2.48 Interfaz cliente-servidor para Modelos Personalizados

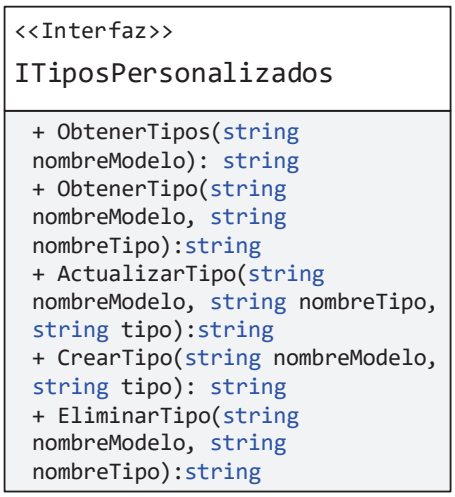


Figura 2.49 Interfaz cliente-servidor para Tipos Personalizados

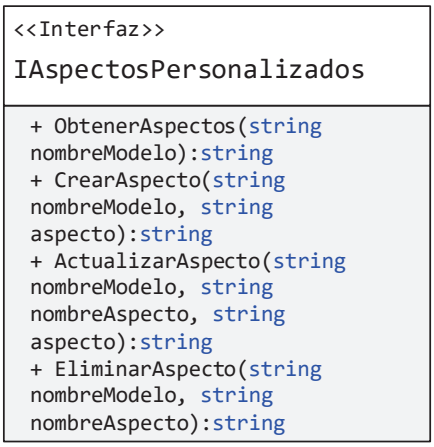


Figura 2.50 Interfaz cliente-servidor para Aspectos Personalizados

Las clases que se utilizaron en este *sprint* son `Node`, `Model`, `Aspect` y `Type`. `Node` y `Aspect` fueron creadas en el *sprint* Visualización de contenidos de usuario por lo que se les agregó lo necesario para que funcionen en este *sprint*. Por otra parte, la clase `Type` y la clase `Model` se diseñaron en este *sprint* y su función es soportar la asociación de objetos (*mapping*) JSON a objetos C#. Cabe recalcar que tal asociación de clases fue manual y no autogenerada mediante un archivo Swagger debido a que la ContentCMN API es una API no oficial de Alfresco.

La relación entre clases que se formó después de diseñar la clase `Type` y `Model` se muestra en el diagrama del ANEXO III. La clase `Type` es muy parecida a la clase `Aspect`, sus atributos se derivaron de los archivos `XML` de Alfresco y del análisis mediante Wireshark de ciertas solicitudes/respuestas que involucran Tipos. Por otra parte, cabe

recalcar que la clase `Aspect` y `Type`, son clases que se diseñaron tanto para gestionar Modelos de Contenido y también se utilizan como atributos de la clase `Nodo`.

2.5.3 Desarrollo

Para este *sprint* se han considerado cuatro formularios: un formulario contenedor (`FGestorModelos`) y tres formularios contenidos (`FModelos`, `FTipos` y `FAspectos`).

El formulario contenedor `FGestorModelos` contiene los métodos para abrir los formularios contenidos y para volver al *dashboard* de usuario.

El formulario `FModelos` es el encargado de desplegar los Modelos de usuario con sus propiedades principales, para ello se ha utilizado el control `datagridview` el cual permite mostrar una tabla. El método `PoblarDtgvModelos` mostrado en el Código 2.12 es el encargado de obtener los modelos personalizados a través de la clase estática `ModelosPersonalizadosStatic` (línea 45) y luego de ello asociar cada campo de cada modelo a uno de la tabla donde se presentarán (líneas 48-51).

```
42      private async Task PoblarDtgvModelos(string filtro)
43      {
44          List<Model> modelos = new List<Model>();
45          modelos = await ModelosPersonalizadosStatic.ObtenerModelosPersonalizados();
46          dtgviewDatos.AutoGenerateColumns = false;
47          dtgviewDatos.DataSource = modelos;
48          dtgviewDatos.Columns["clmNombreModelo"].DataPropertyName = "Name";
49          dtgviewDatos.Columns["clmEspacioNombres"].DataPropertyName = "NamespaceUri";
50          dtgviewDatos.Columns["clmAuthor"].DataPropertyName = "Author";
51          dtgviewDatos.Columns["clmEstadoModelo"].DataPropertyName = "Status";
52      }
```

Código 2.12 Método para mostrar los modelos en una tabla

Para crear un nuevo modelo se utiliza el Código 2.13 el cual crea un nuevo objeto del tipo `Model` (línea 1), agrega sus atributos provenientes de las cajas de texto (líneas 2-5) y a través del método `CrearModeloPersonalizado` de la clase estática `ModelosPersonalizadosStatic` envía el modelo para su creación en el servidor de Alfresco (línea 12). Este fragmento de código pertenece al manejador de evento del botón para actualizar.

Por otra parte, los formularios `FTipos` y `FAspectos` comparten comportamientos similares. Para estos formularios se utilizan `datagridview` para mostrar a manera de lista los Aspectos o los Tipos de usuario, incluso comparten el mismo nombre de columnas descriptoras.

En el Código 2.14 muestra un fragmento de código para la edición de un Tipo, para ello se captura los diferentes campos de las cajas de texto para armar un objeto del tipo `Type`(líneas 2-6),el cual servirá para ser enviado al servidor de Alfresco. El envío se realiza a través del método estático `ActualizarTipoPersonalizado` de la clase `TiposPersonalizadosServicio` (línea 8), si el tipo se ha actualizado de manera correcta entonces se muestra un mensaje de aceptación (líneas 9-10), se actualiza la tabla donde se encuentran los tipos de usuario (línea 11) y se limpian los campos de edición a través del método `NuevaPlantilla` (línea 12). Este fragmento de código pertenece al manejador de evento del botón para actualizar.

```
1  Model modelo = new Model();
2  modelo.Name = txtNombreModelo.Text;
3  modelo.NamespaceUri = txtEspacioModelo.Text;
4  modelo.NamespacePrefix = txtPrefijoModelo.Text;
5  modelo.Description = txtDescripcionModelo.Text;
6  var mdiParent = MdiParent as FDashboard;
7  string autorModelo = PersonasStatic.PersonaAutenticada.FirstName +
8  PersonasStatic.PersonaAutenticada.LastName;
9  modelo.Author = autorModelo;
10 modelo.Status = "DRAFT";
11 fPrincipalLoading.Show();
12 await ModelosPersonalizadosStatic.CrearModeloPersonalizado(modelo);
13 fPrincipalLoading.Close();
14 MessageBox.Show("El nuevo modelo ha sido creado con éxito");
15 await PoblarDtgvModelos("Sin filtro");
16 dtgviewDatos.Refresh();
17 panelModelo.Visible = false;
```

Código 2.13 Creación de un nuevo modelo

```
1  Modelos.CMM.Type tipoActualizar = new Modelos.CMM.Type();
2  tipoActualizar.Name = txtNombre.Text;
3  tipoActualizar.ParentName = cmbxPadre.SelectedItem.ToString();
4  tipoActualizar.Description = txtDescripcion.Text;
5  tipoActualizar.Title = txtTitulo.Text;
6  tipoActualizar.ModeloPerteneciente.Name = modelo.Name;
7  fPrincipalLoading.Show();
8  await TiposPersonalizadosStatic.ActualizarTipoPersonalizado(tipoActualizar);
9  MessageBox.Show("El tipo ha sido actualizado exitosamente.", "Actualizado",
10 MessageBoxButtons.OK, MessageBoxIcon.Information);
11 await PoblarDtgv();
12 NuevaPlantilla();
```

Código 2.14 Edición de un tipo

En el Código 2.15 se muestra parte del código para eliminar un Aspecto a través del manejador del evento clic del menú contextual (opción Eliminar) del `datagridview`. Para ello, primero se muestra una ventana de carga hasta que la acción sea completada (línea 1), luego se ejecuta el método asíncrono `EliminarAspectoPersonalizado` el cual

recibe el nombre del modelo y el nombre del aspecto a eliminarse (línea 2) , por último, si todo ha salido correcto entonces se muestra un mensaje de confirmación (línea 5), se cierra la ventana de carga (línea 4) y se refresca el `datagridview` de Aspectos (línea 7 y 8).

```
1  fPrincipalLoading.Show();
2  await AspectosPersonalizadosStatic.EliminarAspectoPersonalizado(modelo.Name,
3   dtgviewDatos.SelectedRows[0].Cells[0].Value.ToString());
4  fPrincipalLoading.Close();
5  MessageBox.Show("El aspecto ha sido eliminado", "Eliminado", MessageBoxButtons.OK,
6  MessageBoxIcon.Information);
7  await PoblarDtgv();
8  dtgviewDatos.Refresh();
```

Código 2.15 Código para eliminar un aspecto

2.5.4 Entregables

Los entregables de este *sprint* fueron la pantalla para realizar el CRUD de Modelos (ver Figura 2.51), la pantalla para realizar el CRUD de Tipos (ver Figura 2.52) y la pantalla para hacer el CRUD de Aspectos (ver Figura 2.53).

2.5.5 Revisión y Retrospectiva

Los cambios realizados a lo largo de la ejecución de este *sprint* fueron:

1. Agregación de una barra de navegación para poder regresar desde Tipos o Aspectos a la pantalla de Modelos.
2. Validación de los valores ingresados en el campo Nombre de todos los formularios.
3. Cambio en el método de acceso a Propiedades. Se implementó un menú contextual para realizar tal acceso en vez de navegar a través de botones.

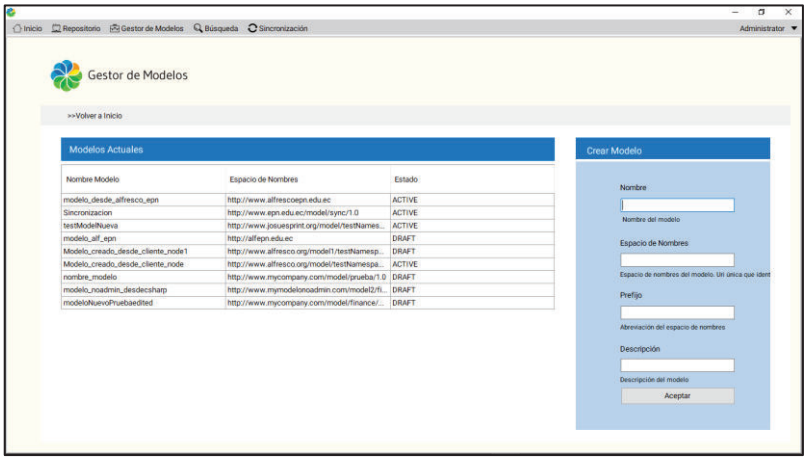


Figura 2.51 Pantalla del CRUD de Modelos del entregable del tercer *sprint*

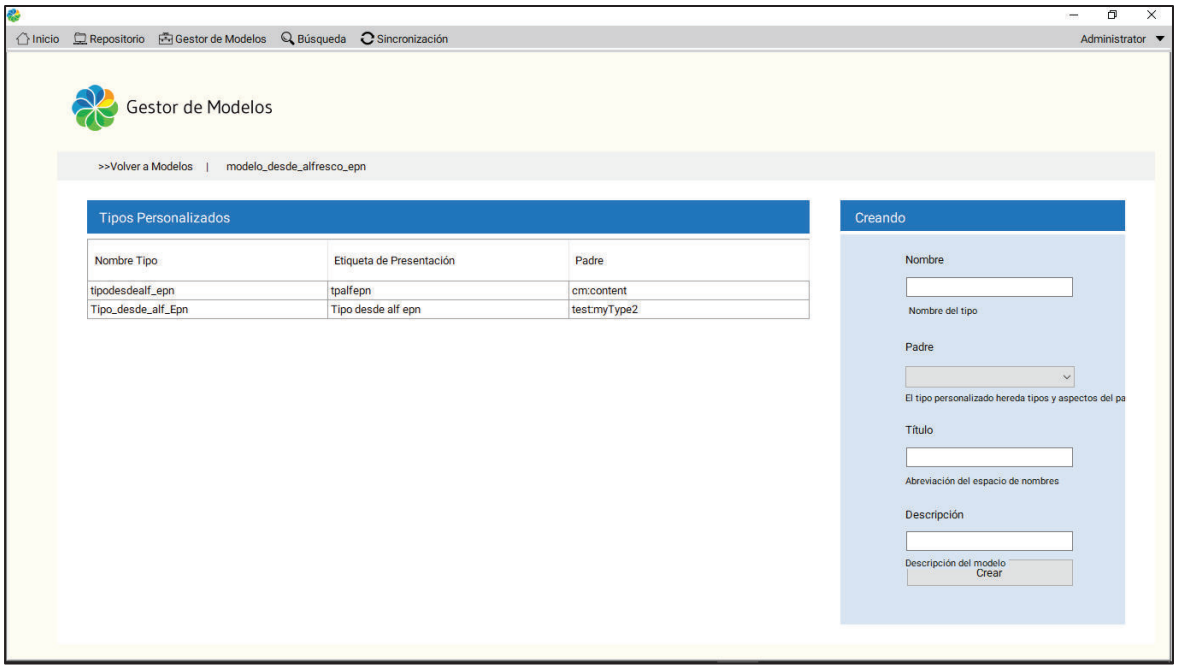


Figura 2.52 Pantalla del CRUD de Tipos del entregable del tercer *sprint*

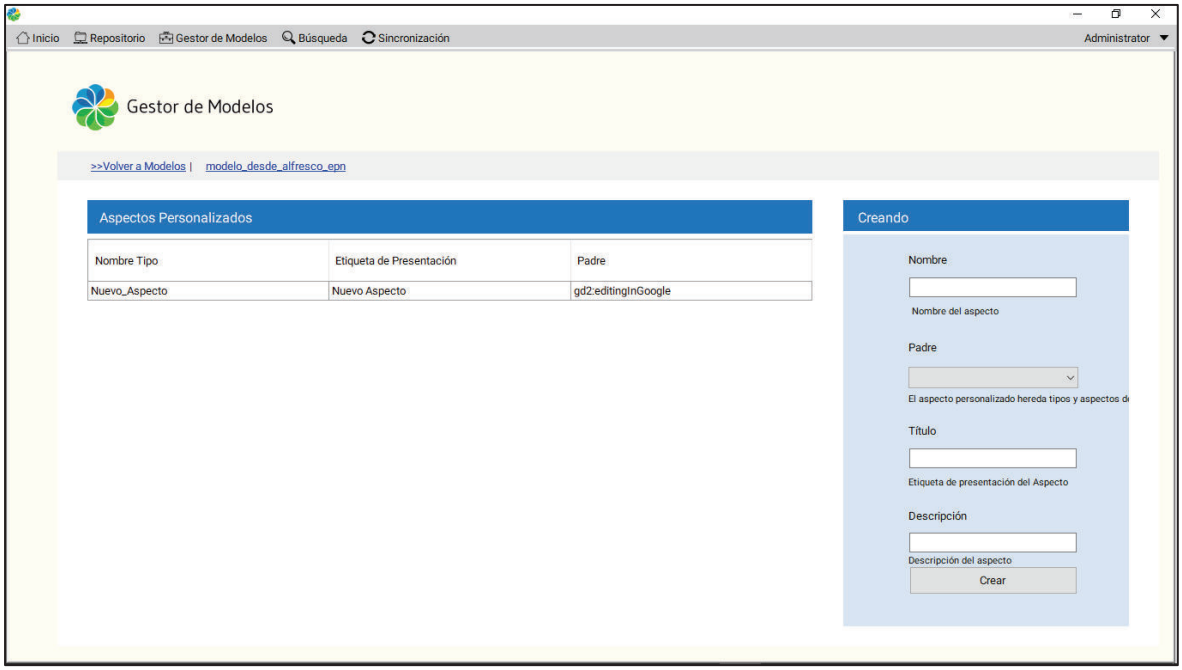


Figura 2.53 Pantalla del CRUD de Aspectos del entregable del tercer *sprint*

2.6 Sprint CRUD de Propiedades

Este *sprint* permite cumplir con el requerimiento para permitir la creación, lectura, edición y eliminación de Propiedades. Este *sprint* ha sido separado del quinto *sprint* debido a su complejidad al momento de realizar el CRUD de una Propiedad.

2.6.1 Interfaz gráfica

Para este *sprint* se ha considerado el *sketch* de la Figura 2.54. Este *sketch* presenta un formulario hijo del *sketch* gestor de Modelos de la Figura 2.40, por lo que conserva la estructura (paneles y botones de navegación de los demás hijos) y se puede acceder desde el formulario de Aspectos o Tipos tal y como se detalla en el *wireframe* de la Figura 2.55.

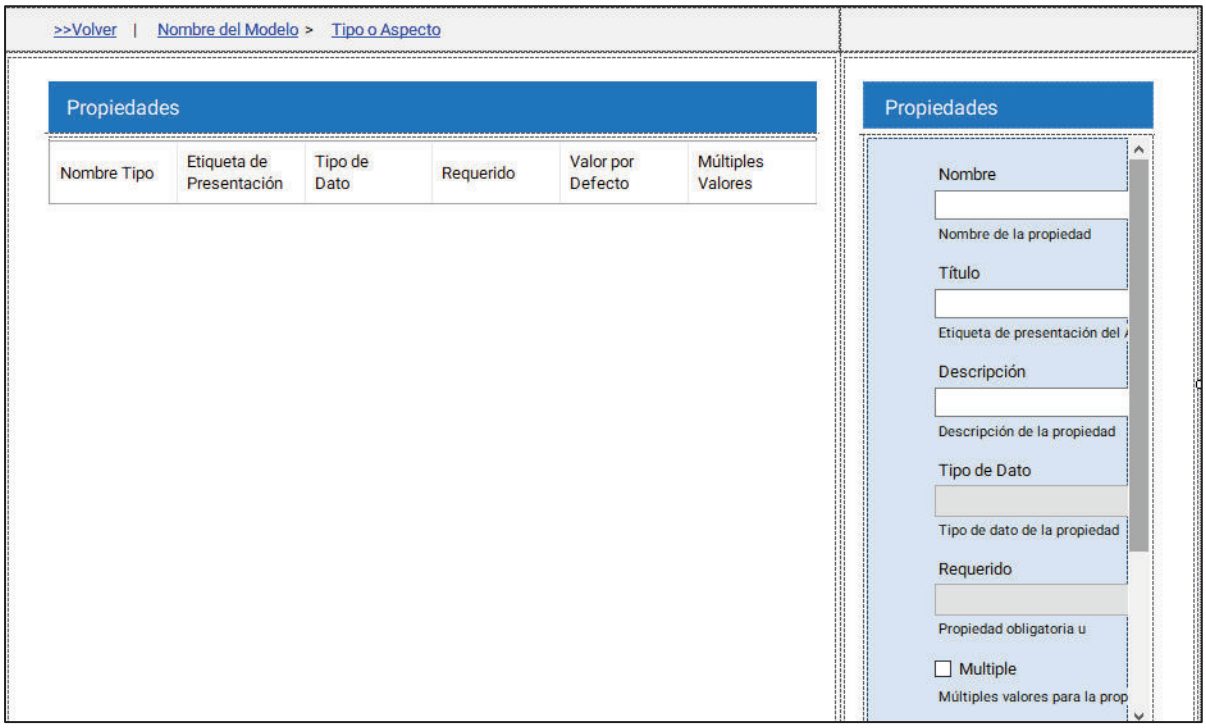


Figura 2.54 *Sketch* para CRUD de propiedades

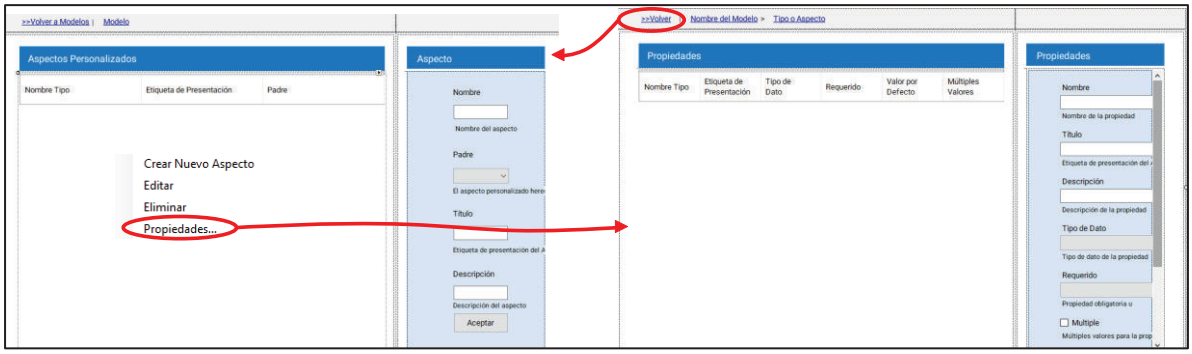


Figura 2.55 *Wireframe* de navegación entre Aspectos/Tipos (izq.) y Propiedades (der.)

2.6.2 Clases e interfaces

Para este *sprint* se han considerado las interfaces de comunicación del *sprint* CRUD de Modelos, Tipos y Aspectos, sin embargo, se han aumentado los métodos necesarios para cumplir las funcionalidades de este *sprint*. En la Figura 2.56, se muestran los métodos

aumentados en la interfaz `ITiposPersonalizados` y en la Figura 2.57 los métodos aumentados a `IAspectosPersonalizados`.

El diagrama de clases que se ha utilizado para este *sprint* se muestra en la Figura 2.59. Los atributos que se ha utilizado para la clase `Property`, `Constraint` y `Parameter` han sido derivados de archivos `xml` de Alfresco, fragmentos de capturas de solicitudes enviadas desde el cliente de Alfresco y de la ContentCMN API. Si bien algunos de los atributos no se utilizan en la implementación, la idea es ser escalables para cualquier otra implementación.

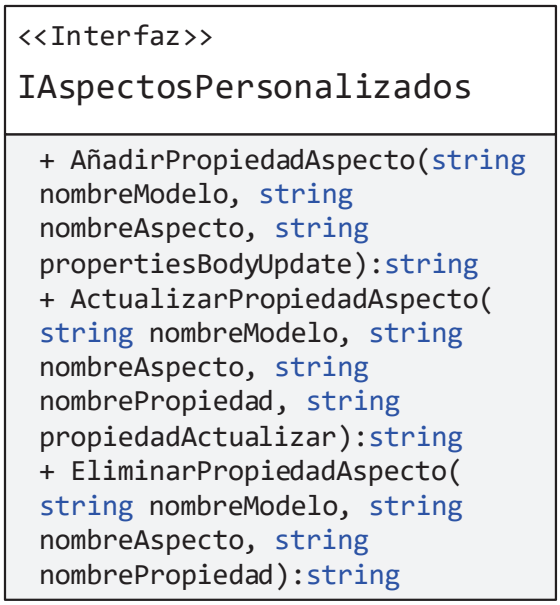


Figura 2.56 Métodos aumentados a la interfaz `IAspectosPersonalizados`

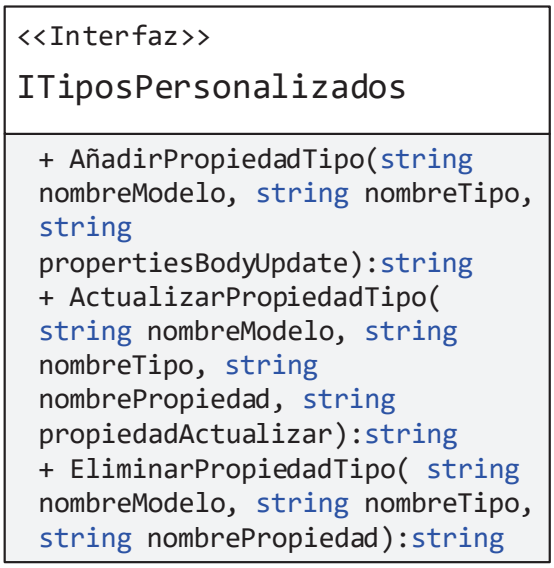


Figura 2.57 Métodos aumentados a la interfaz `ITiposPersonalizados`

Además, se ha creado una clase adicional que permite actualizar propiedades, esta clase se llama `PropertiesBodyUpdate` (Figura 2.58). Esta clase nace del análisis de solicitudes HTTP que ejecuta el cliente por defecto de Alfresco, no está asociada en ningún archivo Swagger sino más bien se propone como una estructura para un futuro archivo Swagger basado en las API no oficiales de Alfresco.

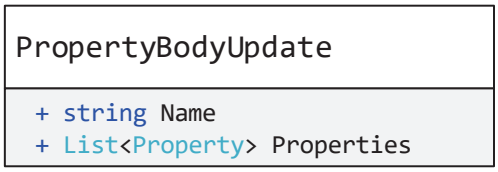


Figura 2.58 Clase para actualizar Propiedades

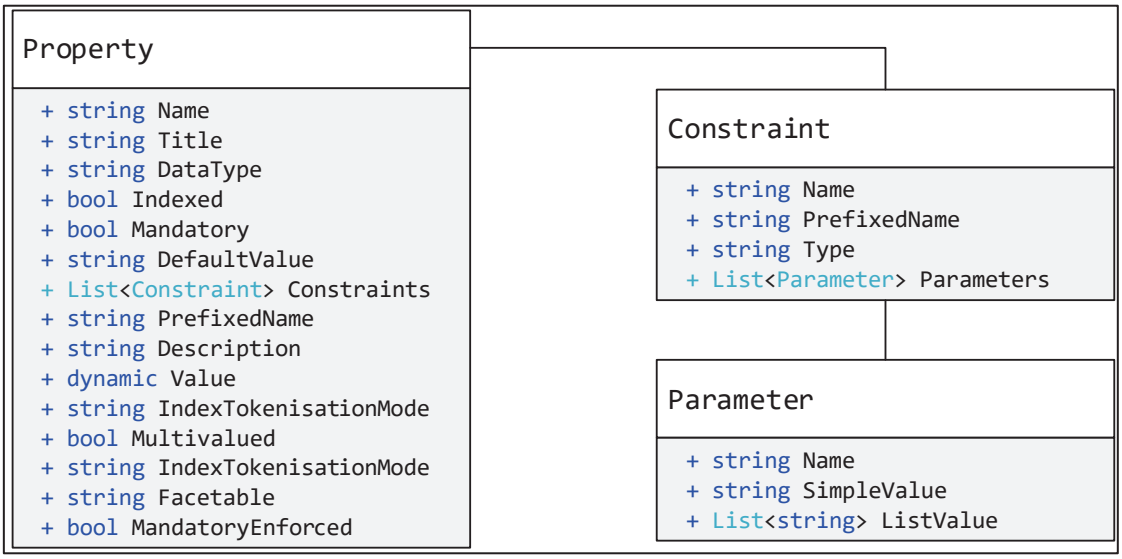


Figura 2.59 Diagrama UML de clases para asociar propiedades JSON a objetos C#

2.6.3 Desarrollo

Para este *sprint* se creó un formulario (`FPropiedades`) el cual contiene un control `datagridview` para mostrar las Propiedades del Tipo o Aspecto, además contiene una barra de navegación, en la parte superior del control `datagridview`, conformada por botones que permiten regresar al Tipo o Aspecto originador, por último este formulario está conformado por un control `flowlayoutpanel` el cual alberga a varios controles del tipo `TextBox` y `ComboBox` que permiten ingresar o mostrar los atributos de alguna propiedad seleccionada desde el control `datagridview`. Para eliminar o editar alguna propiedad se cuenta con un menú contextual sobre el `datagridview`, donde Eliminar elimina el aspecto detrás del menú contextual y el botón Editar carga la propiedad detrás del menú contextual

En el panel para poder cambiar los diferentes atributos de las propiedades seleccionadas. en la Tabla 2.11 se muestra los controles utilizados, sus nombres y su propósito.

Tabla 2.11 Controles accionables utilizados para el CRUD de Propiedades

Nombre	Control	Acción
flwlypanelPropiedades	FlowLayoutPanel	Alberga textbox, combobox y checkbox que representan los valores de los atributos de una Propiedad.
btnAceptar	Button	Crea o Edita una propiedad
dtgviewDatos	DataGridView	Despliega las Propiedades pertenecientes a un Aspecto o Tipo
lnklblVolverNav	LinkLabel	Regresa al formulario anterior
lnklblModeloNav	LinkLabel	Regresa al modelo dueño de la Propiedad
lnklblSubModeloNav	LinkLabel	Regresa al Tipo o Aspecto de la Propiedad
txtNombre	TextBox	Captura o despliega el nombre de la Propiedad
txtTitulo	TextBox	Captura o despliega el título de la Propiedad
txtDescripcion	TextBox	Captura o despliega la descripción de la Propiedad
cmbxTipoDato	ComboBox	Captura o despliega el tipo de dato de la Propiedad
cmbxRequerido	ComboBox	Captura o despliega si la Propiedad es requerida o no
chkboxMultiple	CheckBox	Captura o despliega si la Propiedad contiene múltiples valores
cmbxRestriccion	ComboBox	Captura o despliega si la Propiedad tiene alguna restricción

El Código 2.16 presenta cómo crear una Propiedad. Las líneas 93 a 109 muestran cómo se crea un objeto del tipo `Property` y se le añade sus atributos provenientes de los controles `TextBox` de la interfaz de usuario. Otraz propiedades se derivan de la selección del control `ComboBox`, como por ejemplo las líneas 105-108, son agregadas en caso de que la selección del control `ComboBox cmbxIndexacion` sea igual al valor `Ninguno`. Una vez que el objeto ha sido inicializado, para poder agregarlo a un Aspecto o Tipo es necesario crear una lista de objetos del tipo `Property` (línea 111), agregar la propiedad creada a tal lista (línea 112) y enviar tal lista como atributo del objeto `PropertiesBodyUpdate` (línea 113). Este objeto se lo envía como parámetro de entrada al método `AñadirPropiedadesAspecto` (conjuntamente con el nombre del Tipo o Aspecto proveniente), el cual es el encargado de realizar la solicitud HTTP que inserta una propiedad sobre un Tipo o Modelo.

```
93              Property propiedadCrear = new Property();
94              propiedadCrear.Name = txtNombre.Text;
95              propiedadCrear.Description = txtDescripcion.Text;
96              propiedadCrear.Title = txtTitulo.Text;
97              propiedadCrear.Datatype = cmbxTipoDato.SelectedItem.ToString();
98              propiedadCrear.MultiValued = chkboxMultiple.Checked;
99              if (cmbxRequerido.SelectedItem.ToString() == "Opcional") propiedadCrear.Mandatory = false;
100             else { propiedadCrear.Mandatory = true; }
101             if (cmbxRequerido.SelectedItem.ToString() == "Ninguno") propiedadCrear.Constraints = null;
102             else { }
103             if (cmbxIndexacion.SelectedItem.ToString() == "Ninguno")
104             {
105                 propiedadCrear.Facetable = "UNSET";
106                 propiedadCrear.IndexTokenisationMode = "TRUE";
107                 propiedadCrear.Indexed = false;
108                 propiedadCrear.MandatoryEnforced = false;
109             }
110
111             List<Property> propiedadesCrear = new List<Property>();
112             propiedadesCrear.Add(propiedadCrear);
113             PropertiesBodyUpdate propertiesBodyCreate = new PropertiesBodyUpdate(subModelo.Name, propiedadesCrear);
114             if (proveniente == "ASPECTOS")
115             {
116                 await AspectosPersonalizadosStatic.AñadirPropiedadeAspecto(
117                     modelo.Name,
118                     subModelo.Name,
119                     propertiesBodyCreate);
120                 MessageBox.Show("Propiedad creada exitosamente");
121             }
```

Código 2.16 Código para crear una propiedad

2.6.4 Entregables

El único entregable de este *sprint* es la pantalla del CRUD de Propiedades mostrado en la Figura 2.60.

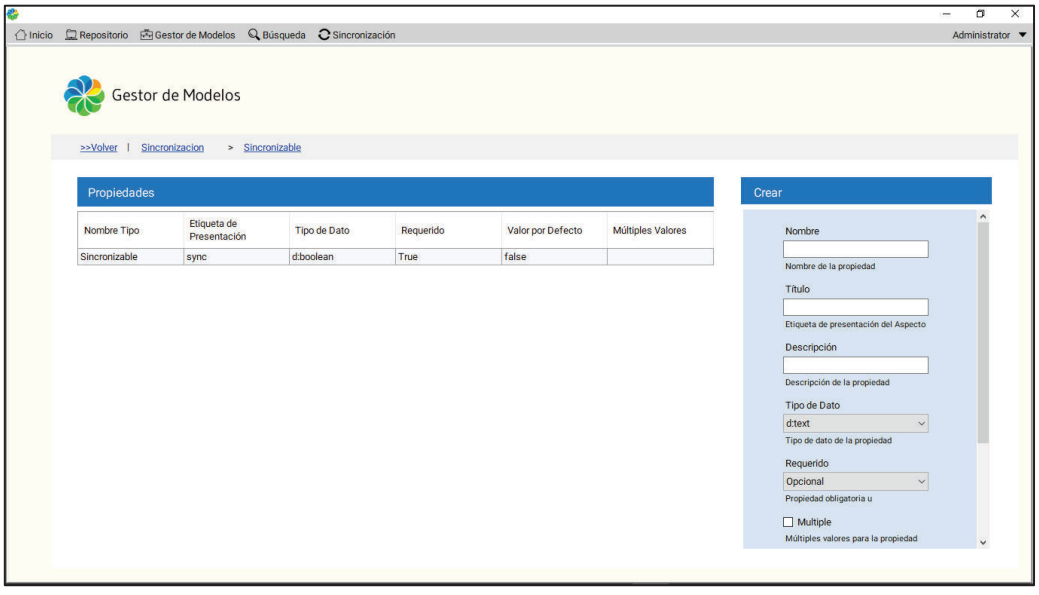


Figura 2.60 Pantalla del CRUD de Propiedades del cuarto *sprint*

2.6.5 Revisión y Retrospectiva

Los cambios realizados a lo largo de la ejecución de este *sprint* fueron:

1. Validación de algunos atributos de la propiedad dependiendo del tipo de dato seleccionado. El tipo de dato determina las diferentes Restricciones que se puden utilizar.

2. Agregación de una caja de texto para poder ingresar fechas en caso de que el tipo de dato especificado sea `Date` o `DateTime`.

3. En caso de que el tipo de dato sea númerico entonces se evitó que sea posible ingresar números en el valor predeterminado de la propiedad, de igual manera se lo realizó con los tipos `float` y `double`, y equivalentemente con el tipo `bool`.

2.7 Sprint Sincronización de Contenido

Este *sprint* se realizó con el objetivo de generar un entregable suficiente para cumplir los requerimientos planteados en el levantamiento de la información: agregar y almacenar de sub-repositorios de Alfresco en la PC y permitir la sincronización de los contenidos de Alfresco con la PC.

2.7.1 Interfaz gráfica

Para este *sprint* se ha considerado un solo *sketch* el cual se muestra en la Figura 2.61. El botón 'Sincronizar' servirá para sincronizar el contenido del sub-repositorio seleccionado del control `listbox` con el contenido de Alfresco o viceversa. El botón 'Agregar Repositorio' servirá para visualizar el control `treeview`. El botón 'Aceptar' servirá para sincronizar por primera vez un sub-repositorio nuevo del control `treeview`.

Figura 2.61 *Sketch* para sincronización de contenido

Al *sketch* de la Figura 2.61 se accede desde el botón “Sincronización” del *dashboard* de usuario o desde el menú de opciones, opción “Sincronización” tal como se muestra en el *wireframe* la Figura 2.62.

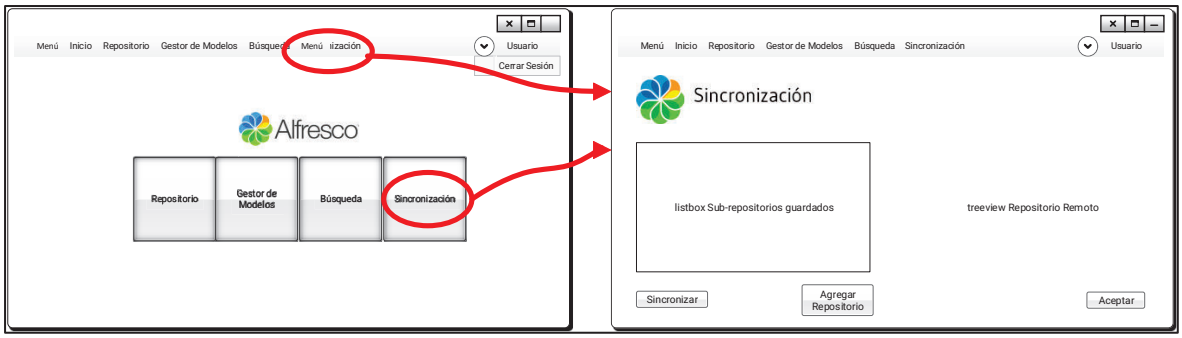


Figura 2.62 *Wireframe* para acceder al módulo de Sincronización

Además, el sketch de la Figura 2.61 tiene relación directa con la carpeta sincronizada de la computadora de usuario, debido a que el botón “Sincronizar” modifica, crea o elimina de ser necesario los archivos o carpetas de una carpeta local. Esta interacción se muestra en el *wireframe* de la Figura 2.63. A su vez, el botón “Sincronizar” modifica, crea o elimina los archivos o carpetas del repositorio Alfresco (remoto) según los cambios que hayan sufrido sus símiles de la carpeta local.

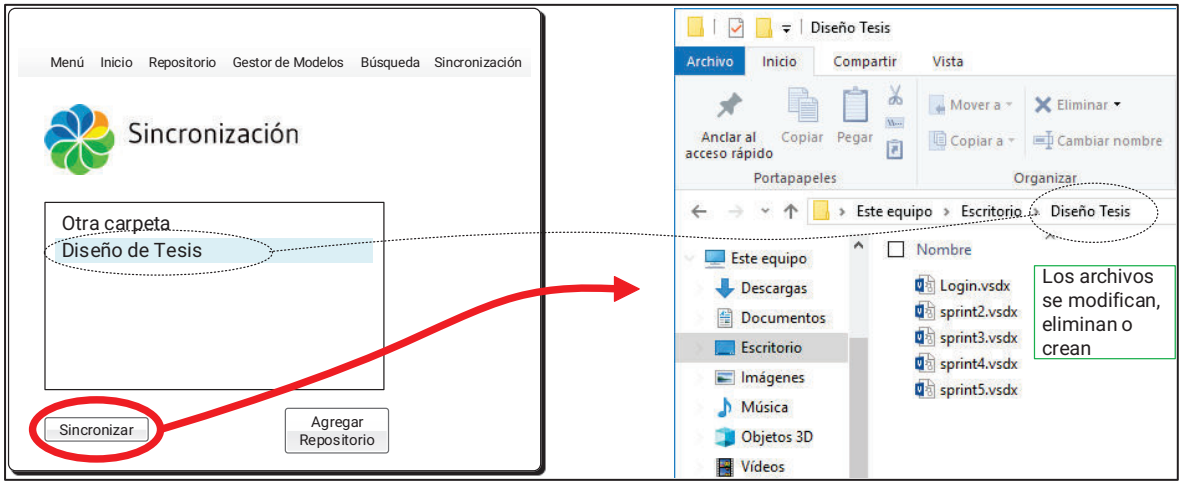


Figura 2.63 *Wireframe* de interacción entre módulo Sincronización y carpeta local

2.7.2 Clases e interfaces

Antes de considerar las clases encargadas de manejar los objetos necesarios para satisfacer el requisito de sincronización, se realizó un análisis de sincronización (Tabla 2.12). En la Figura 2.64 se muestra que para lograr una sincronización completa es necesario realizar comparaciones desde la PC hacia Alfresco y desde Alfresco hacia la PC,

las acciones que se realizan en cada comparación se detallan en la Tabla 2.12. Estas acciones requieren de algunos metadatos asociados tanto al servidor como al cliente que se detallarán a continuación.

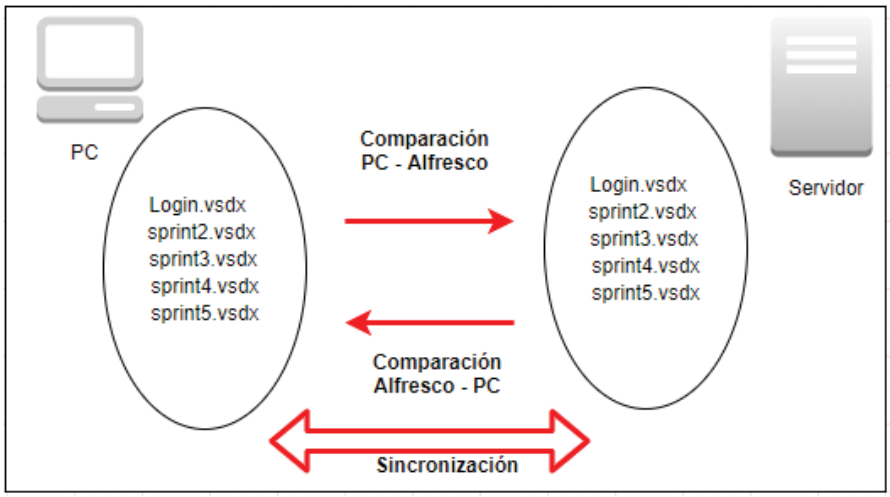


Figura 2.64 Análisis para sincronización entre PC y Alfresco

Para este *sprint* se han considerado las clases `Nodo` y `NodoLocal`. La clase `Nodo` se la creó en el *sprint* Visualización de contenidos de usuario, por lo que se le agregó lo necesario para satisfacer las funcionalidades de este *sprint*. Por otra parte, la clase `NodoLocal` se la creó como representación de un archivo o carpeta en la computadora del usuario. `NodoLocal` y sus atributos se muestran en el diagrama de clases UML de la Figura 2.65. `NodoLocal` es capaz de almacenar datos sobre los archivos de la PC a través de métodos que permiten la lectura, agregación y edición de metadatos sobre los archivos y carpetas de la PC.

Tabla 2.12 Comparación requerida para cada acción de usuario

Caso de uso	Comparación requerida
El usuario crea un archivo/carpeta en la computadora	Computador-Alfresco
El usuario modifica un archivo/carpeta en la computadora	En ambos sentidos
El usuario elimina un archivo/carpeta en la computadora	Alfresco-Computador
El usuario crea un nodo en Alfresco	Alfresco-Computador
El usuario modifica un nodo en Alfresco	En ambos sentidos
El usuario elimina un nodo en Alfresco	Computador-Alfresco

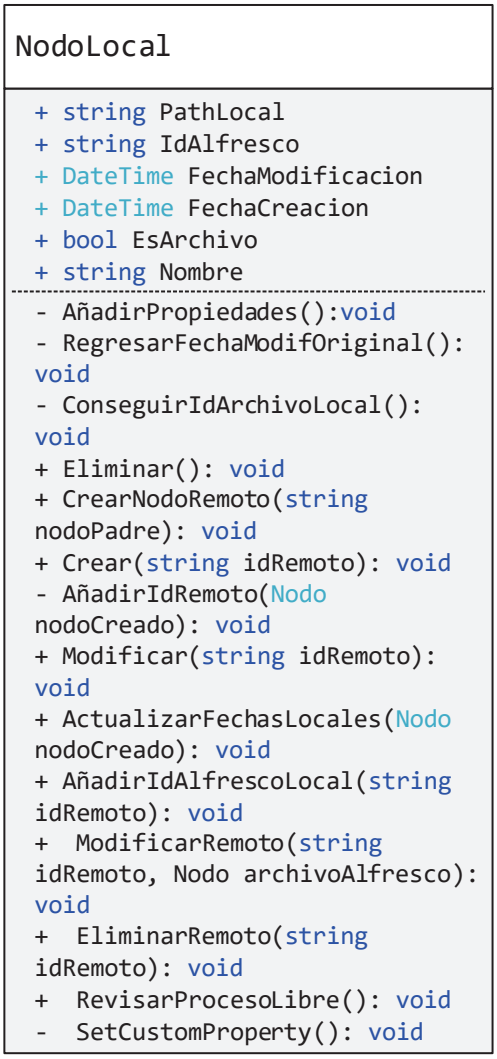


Figura 2.65 Clase NodoLocal

`NodoLocal` cuenta con métodos que permiten almacenar metadatos en los archivos o carpetas locales. Se definió el diagrama E-R (Figura 2.66) que permite guardar localmente atributos de un Nodo.

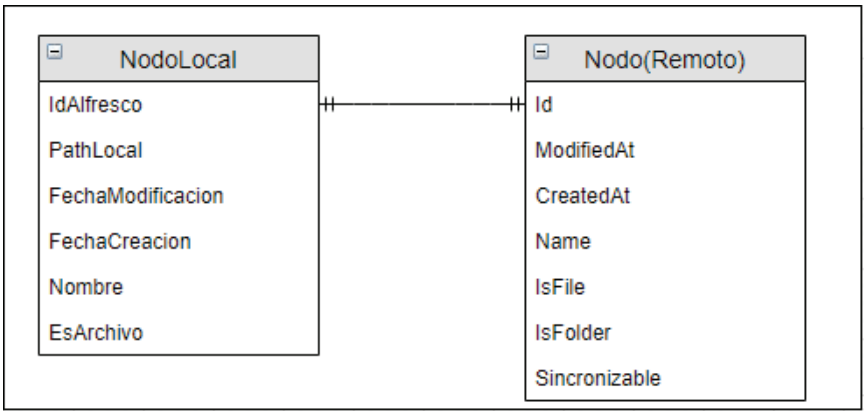


Figura 2.66 Diagrama E-R para almacenamiento de Nodos locales

Por otra parte, en este *sprint* también se decidió guardar algunos parámetros de configuración del prototipo. En la Figura 2.67 se muestran los parámetros que han sido agregados, estos parámetros son utilizados para guardar la ubicación e identificar los sub-repositorios guardados por el usuario en la PC.

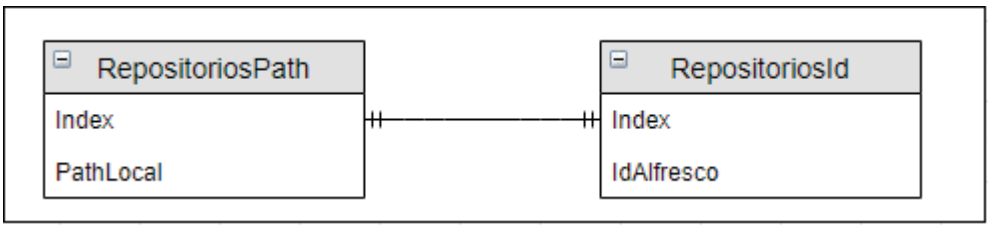


Figura 2.67 Diagrama E-R para almacenamiento de sub-repositorios

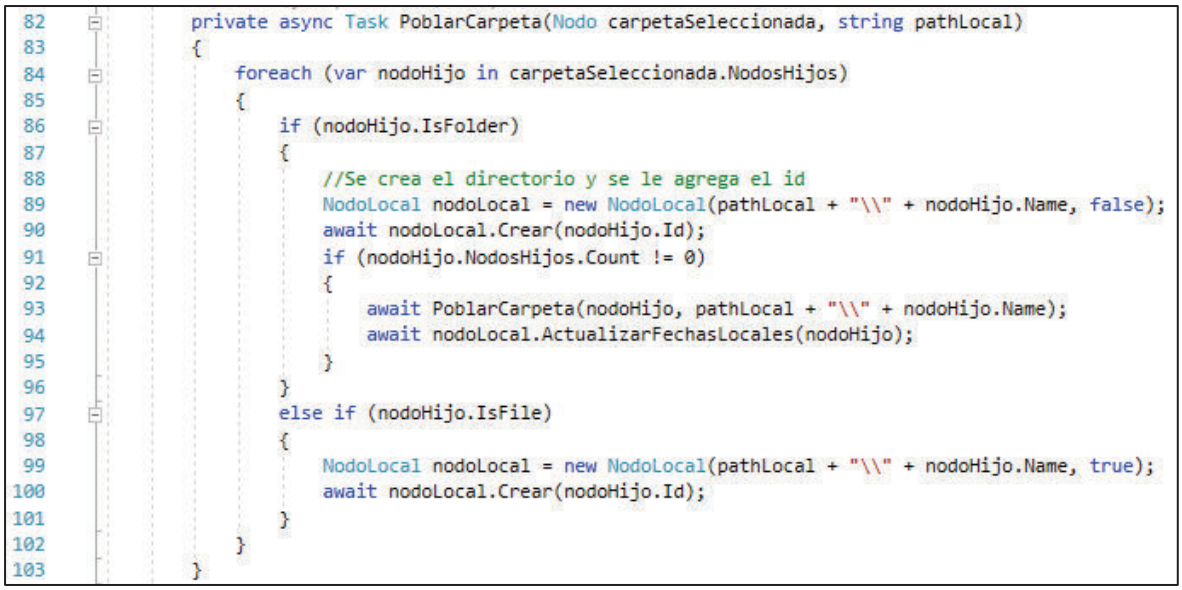
2.7.3 Desarrollo

Para la interfaz gráfica de este *sprint* se han considerado los controles indicados en la Tabla 2.13.

Tabla 2.13 Controles accionables utilizados para Sincronización

Nombre	**Control**	**Acción**
`lstboxRepositoriosGuardados`	`ListBox`	Muestra una lista de los repositorios guardados
`btnSincronizar`	`Button`	Sincroniza el sub-repositorio local seleccionado, con el servidor de Alfresco
`btnAgregarRepo`	`Button`	Permite seleccionar y agregar un sub-repositorio a la lista de repositorios guardados
`btnAceptar`	`Button`	Permite agregar un sub-repositorio a la lista de repositorios guardados
`treeviewSync`	`TreeView`	Permite seleccionar un sub-repositorio a la lista de repositorio guardados

Para descargar el contenido de una carpeta remota en una ubicación local, se utilizó el método `PoblarCarpeta` el cual se muestra en el Código 2.17; allí se utiliza el constructor de la clase `NodoLocal` para inicializar la ubicación local del nodo local (línea 89) y si es archivo o carpeta, además si el nodo es carpeta se realizará una llamada recursiva al método `PoblarCarpeta` para que el contenido de esa carpeta sea descargado, también se actualizan las fechas de modificación y de creación de los nodos debido a que cuando se descargan se vuelven a crear y no contienen la última fecha de modificación del Nodo Remoto (líneas 91-95).



```
82          private async Task PoblarCarpeta(Nodo carpetaSeleccionada, string pathLocal)
83          {
84              foreach (var nodoHijo in carpetaSeleccionada.NodosHijos)
85              {
86                  if (nodoHijo.IsFolder)
87                  {
88                      //Se crea el directorio y se le agrega el id
89                      NodoLocal nodoLocal = new NodoLocal(pathLocal + "\\" + nodoHijo.Name, false);
90                      await nodoLocal.Crear(nodoHijo.Id);
91                      if (nodoHijo.NodosHijos.Count != 0)
92                      {
93                          await PoblarCarpeta(nodoHijo, pathLocal + "\\" + nodoHijo.Name);
94                          await nodoLocal.ActualizarFechasLocales(nodoHijo);
95                      }
96                  }
97                  else if (nodoHijo.IsFile)
98                  {
99                      NodoLocal nodoLocal = new NodoLocal(pathLocal + "\\" + nodoHijo.Name, true);
100                     await nodoLocal.Crear(nodoHijo.Id);
101                 }
102             }
103         }
```

Código 2.17 Método `PoblarCarpeta`. Descarga recursivamente nodos remotos

En el Código 2.18 se muestra un fragmento del método `SincronizaciónPcAlfresco`, el cual se encarga de buscar cada nodo local en la lista descargada de nodos remotos, pertenecientes a un mismo directorio, luego a través del método `VerificarPath` (línea 158) se verifica que el nodo local pertenezca a la carpeta actual, debido a que si un nodo (ya sincronizado) es copiado por el usuario utilizando el Explorador de Archivos, el nodo copiado conserva los metadatos de su anterior ubicación y es necesario reemplazar tales metadatos.

Si la búsqueda del nodo de la PC en la lista de nodos remotos (líneas 166-170), pertenecientes al mismo directorio, es insatisfactoria (devuelve `null`) y el nodo local ya tiene asignado un `id`, entonces quiere decir que el nodo local se ha eliminado de Alfresco y es necesario eliminarlo en la PC también.

Si la búsqueda del nodo de la PC en la lista de nodos remotos (líneas 172-175), pertenecientes al mismo directorio, es insatisfactoria (devuelve `null`) y el nodo local no tiene asignado un `id`, entonces quiere decir que el nodo local se ha creado en la PC por lo que es necesario crearlo en Alfresco también.

Si la búsqueda del nodo de la PC en la lista de nodos remotos, pertenecientes al mismo directorio, es satisfactoria (devuelve un `Nodo`) (Líneas 177-184) y las fechas de modificación del nodo local y del nodo remoto no coinciden entonces quiere decir que el nodo local se ha modificado en Alfresco o en la PC. Entonces habrá que volver a cargar el nodo local en la PC y en Alfresco.

```
152             foreach (var nodoLocal in nodosLocales)
153             {
154                 Nodo archivoAlfresco = nodoSync.NodosHijos.Find(x => x.Id == nodoLocal.IdAlfresco);
155
156                 if (nodoLocal.IdAlfresco != "")
157                 {
158                     if (!await VerificarPath(nodoLocal))
159                     {
160                         nodoLocal.IdAlfresco = "";
161                         nodoLocal.AñadirIdAlfrescoLocal("");
162                     }
163                 }
164
165                 //Si se ha eliminado en Alfresco, entonces:
166                 if (archivoAlfresco is null && (!nodoLocal.IdAlfresco.Equals("")))
167                 {
168                     nodoLocal.Eliminar();
169                     continue;
170                 }
171                 //Si se ha creado en PC, entonces:
172                 else if ((archivoAlfresco is null && nodoLocal.IdAlfresco.Equals("")))
173                 {
174                     await nodoLocal.CrearNodoRemoto(nodoSync.Id);
175                 }
176                 //Modificación:
177                 else if (archivoAlfresco.ModifiedAt == nodoLocal.FechaModificacion)
178                 {
179                     //En caso de cambio de nombre
180                     if (archivoAlfresco.Name != nodoLocal.Nombre)
181                     {
182                         await nodoLocal.ModificarRemoto(archivoAlfresco.Id, archivoAlfresco);
183                     }
184                 }
```

Código 2.18 Fragmento del método `SincronizaciónPcAlfresco`

La sincronización en el lado del cliente está apoyada por los metadatos almacenados en los archivos locales. En el lado del servidor también se cuenta con un metadato para la sincronización el cual es `Sincronizable` del tipo `bool` y perteneciente al Aspecto `Sincronizable`. A través de este metadato es posible determinar con exactitud cuando un archivo ha sido eliminado del PC. Sin este metadato se tendría la ambigüedad entre las acciones, eliminar en la PC y crear en Alfresco. En el Código 2.19 se muestra el método `SincronizarAlfrescoPc` el cual permite comparar cada nodo remoto contra los nodos locales.

Si la búsqueda de un nodo remoto es insatisfactoria (líneas 270-273), pero el nodo remoto ha sido sincronizado anteriormente, entonces quiere decir que el nodo ha eliminado de la PC, por lo que se elimina al nodo remoto.

Si la búsqueda de un nodo remoto es insatisfactoria (líneas 275-286) y el nodo remoto no ha sido sincronizado anteriormente, entonces quiere decir que el nodo ha creado en Alfresco por lo que se crea (réplica del remoto) un nodo local dependiendo si este es archivo o carpeta (líneas 277-285).

```
255         private async Task SincronizarAlfrescoPc(string repositorioSeleccionado, Nodo nodoSync,
256             List<NodoLocal> nodosLocales)
257         {
258             //*********Comparación desde Alfresco hacia PC*********
259             foreach (var nodoRemoto in nodoSync.NodosHijos)
260             {
261                 NodoLocal nodoLocal = nodosLocales.Find(x => x.IdAlfresco == nodoRemoto.Id);
262                 bool creadoRemotamente = false;
263                 List<string> aspectosNodo = (List<string>)nodoRemoto.AspectNames;
264                 if (!(aspectosNodo.Contains("sync:Sincronizable")))
265                 {
266                     creadoRemotamente = true;
267                 }
268 
269                 //Si se ha eliminado en PC, entonces:
270                 if (nodoLocal is null && !creadoRemotamente)
271                 {
272                     await NodoLocal.EliminarRemoto(nodoRemoto.Id);
273                 }
274                 //Si se ha creado en Alfresco, entonces:
275                 else if (nodoLocal is null && creadoRemotamente)
276                 {
277                     if (nodoRemoto.IsFile)
278                     {
279                         nodoLocal = new NodoLocal(repositorioSeleccionado + "\\" + nodoRemoto.Name, true);
280                     }
281                     else
282                     {
283                         nodoLocal = new NodoLocal(repositorioSeleccionado + "\\" + nodoRemoto.Name, false);
284                     }
285                     await nodoLocal.Crear(nodoRemoto.Id);
286                 }
287             }
```

Código 2.19 Método `SincronizarAlfrescoPc`

En este *sprint* también se implementó la clase `NodoLocal` la cual permite gestionar metadatos en los nodos locales. Para ello se utilizó métodos que permitan crear, modificar, leer y eliminar metadatos en archivos locales. En el Código 2.20 se muestra un fragmento de código el cual permite leer el metadato `IdAlfresco` de un archivo o carpeta. En la línea se inicializa el objeto a través del cual se realizará la gestión de metadatos sobre archivos y carpetas. En la línea 3 se utiliza el método `CustomProperties.Add` el cual permite añadir el metadato deseado.

```
OleDocumentProperties myFile = new DSOFile.OleDocumentProperties();
myFile.Open(PathLocal, false, DSOFile.dsoFileOpenOptions.dsoOptionDefault);
myFile.CustomProperties.Add("IdAlfresco", idRemoto);
myFile.Save();
myFile.Close(true);
```

Código 2.20 Lectura del metadato local IdAlfresco

Para archivos generados con la *suite* de Office el Código 2.20 no es compatible, por lo que para esos archivos se utilizó el método `SetCustomProperty` basado en [37] y modificado para manejar archivos derivados de Power Point y Excel. En el Código 2.21 se muestra un fragmento de código para lectura del metadato `IdAlfresco` de un archivo Office. En las líneas 1-5 se inicializa un *array* que contiene las extensiones de los archivos Office. Si la

extensión del archivo al que se va a agregar metadatos coincide con las extensiones especificadas en el *array* (línea 6) entonces se hará uso del método `SetCustomProperty` (línea 7) para agregar el metadato deseado.

```
1  List < string > extOfficeDocs = new List < string > {
2   ".docx", ".docm", ".dotx", ".dotm", ".docb", ".xlsx",
3   ".xlsm", ".xltx", ".xltm", ".pptx", "pptm", ".potx",
4   "potm", ".ppam", ".ppsm", ".sldx", ".sldm"
5  };
6  if (extOfficeDocs.Contains(Path.GetExtension(PathLocal))) {
7   SetCustomProperty(PathLocal, "IdAlfresco", idRemoto, PropertyTypes.Text, Path.GetExtension(PathLocal));
8  }
```

Código 2.21 Lectura del metadato local `IdAlfresco` (Para archivos Office)

2.7.4 Entregables

El entregable de este *sprint* se muestra en la Figura 2.68.

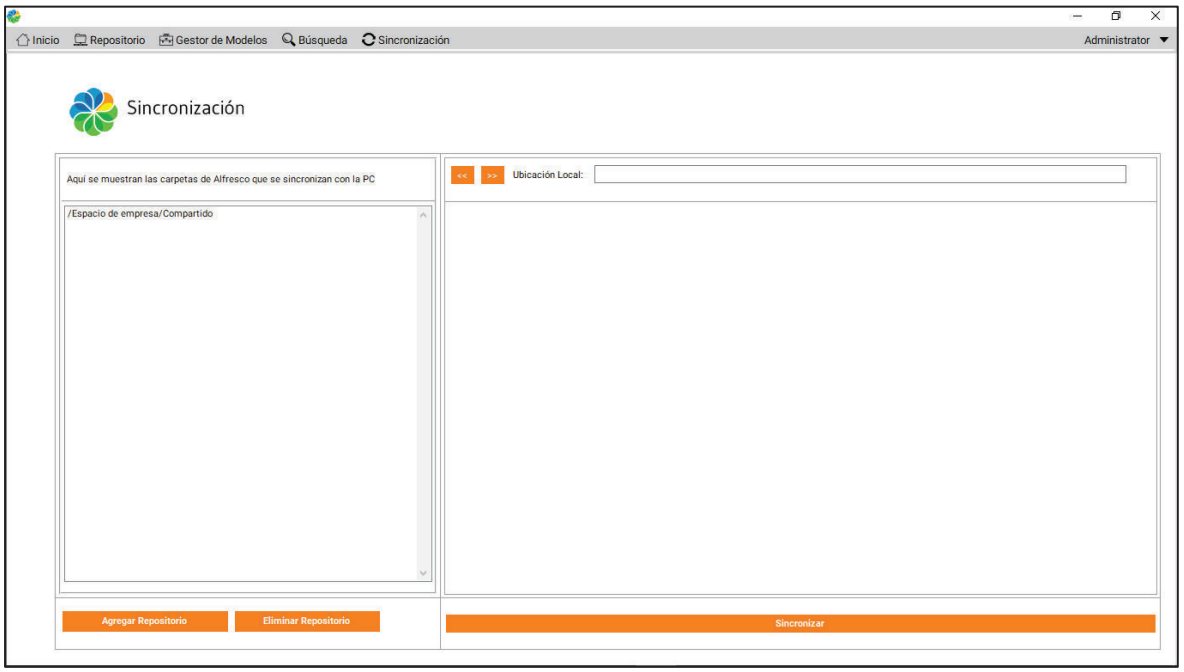


Figura 2.68 Pantalla para sincronización del quinto *sprint*

2.8 Sprint Búsqueda de Contenido basado en metadatos

Este *sprint* se realiza con el fin de generar un entregable que cumpla los requerimientos: agregar y almacenar de sub-repositorios de Alfresco en la PC, permitir la sincronización de los contenidos de Alfresco con la PC, permitir la búsqueda de contenidos basada en tipos, aspectos, propiedades y valores de las propiedades. Al final se tendrá una interfaz que posea varios filtros programables por el usuario que devuelvan los nodos buscados por un determinado criterio de búsqueda.

2.8.1 Diseño

Para la realización de este *sprint* se diseñó un *sketch* (Figura 2.69) que contiene un botón de búsqueda para las diferentes opciones seleccionadas a través de los controles `combobox`, `textbox` o `dateTimePicker`.

La Figura 2.69 muestra el *sketch* del gestor de búsquedas. El orden de selección y búsqueda se refleja en el diagrama de flujo de la Figura 2.70; ese diagrama de flujo indica que es posible buscar Nodos por:

- Tipo
- Aspecto
- Propiedad
- Valor de la Propiedad

Mientras que las operaciones que se pueden aplicar para la búsqueda por valor dependen del tipo de dato y del número de operadores que se decida usar. La Tabla 2.14 muestra las operaciones que se pueden aplicar a la búsqueda por valor.

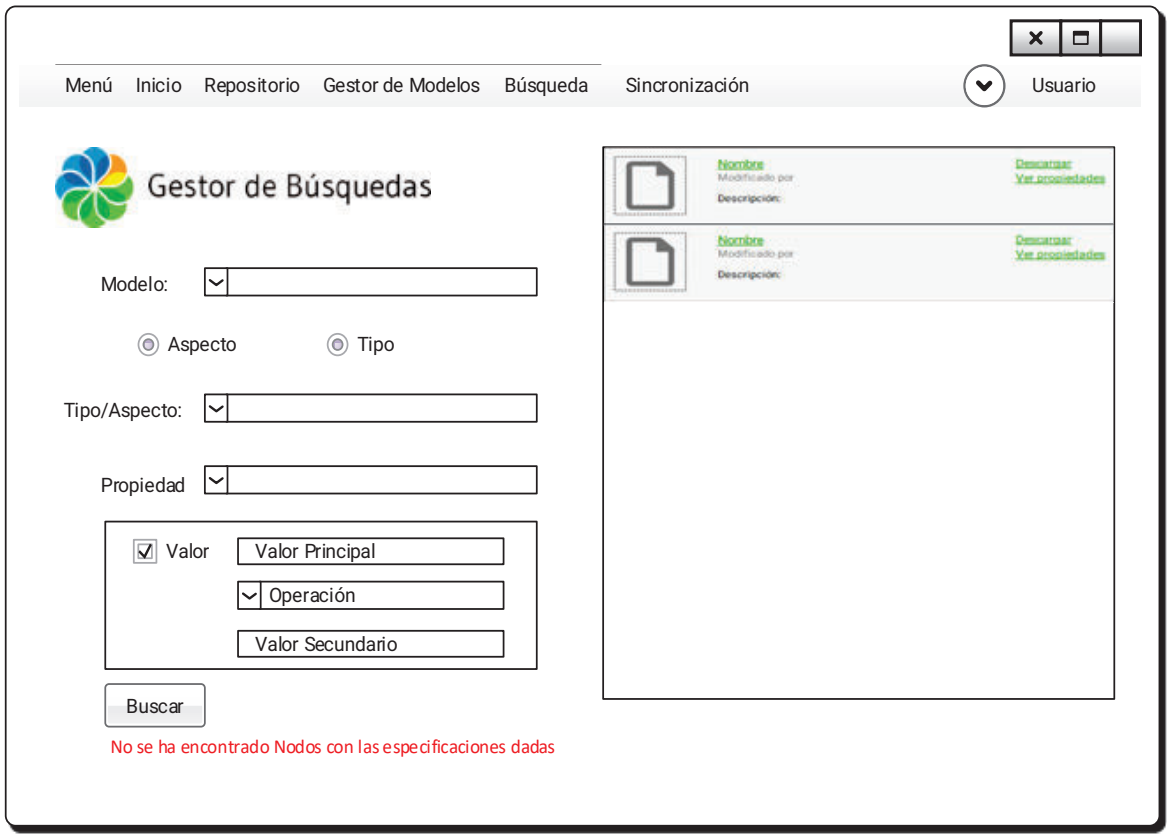


Figura 2.69 *Sketch* del gestor de búsquedas

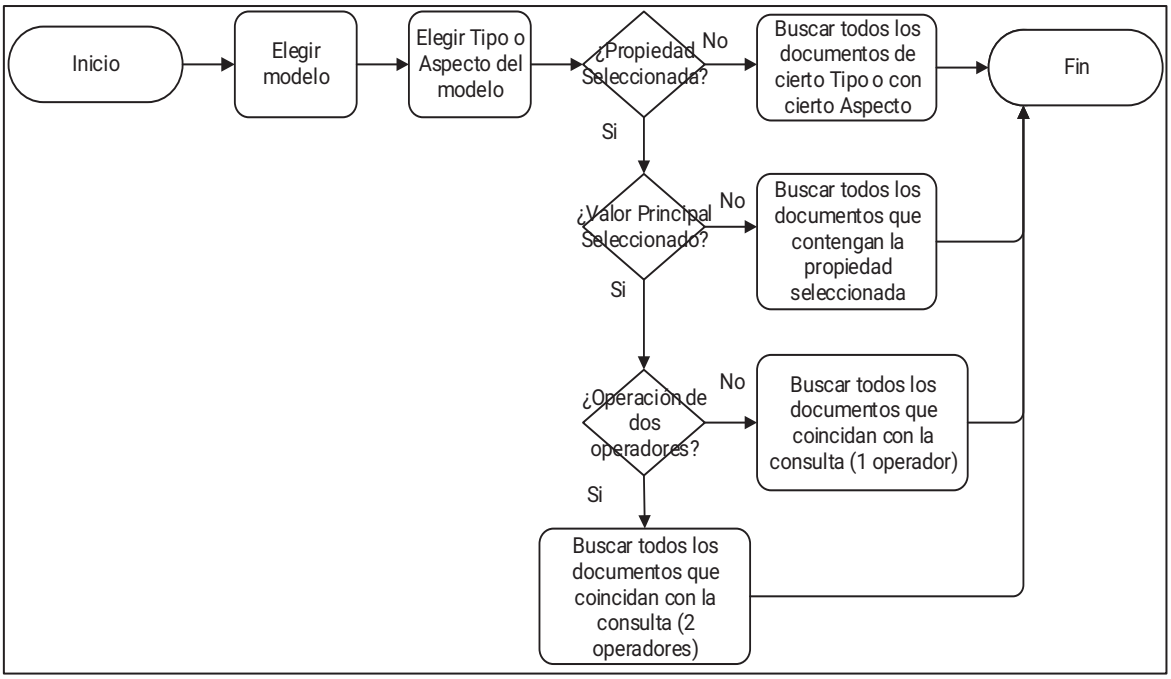


Figura 2.70 Diagrama de Flujo de selección de las opciones de búsqueda

Tabla 2.14 Operaciones aplicadas a la búsqueda por valor

Tipo de dato	**1 operación**	**2 operaciones**
`text`	coincidencia exacta, coincidencia + corrector ortográfico[50]	-
`mltext`	coincidencia exacta, coincidencia + corrector ortográfico	-
`datetime`	mayor que, menor que, mayor o igual que, menor o igual que	rango[][51], rango(), rango[), rango(]
`date`	mayor que, menor que, mayor o igual que, menor o igual que	rango[], rango(), rango[), rango(]
`int`	mayor que, menor que, mayor o igual que, menor o igual que, igual	rango[], rango(), rango[), rango(]
`float`	mayor que, menor que, mayor o igual que, menor o igual que, igual	rango[], rango(), rango[), rango(]
`double`	mayor que, menor que, mayor o igual que, menor o igual que, igual	rango[], rango(), rango[), rango(]
`long`	mayor que, menor que, mayor o igual que, menor o igual que, igual	rango[], rango(), rango[), rango(]
`bool`	coincidencia exacta	-

Para llegar al formulario de Gestión de Búsqueda es necesario pasar por el *dashboard* y a través de la opción de búsqueda, tanto del menú de opciones como de los botones de

[50] Corrector ortográfico: Busca posibles coincidencias de una palabra con error ortográfico.
[51] Rango: Los corchetes representan un rango inclusivo y los paréntesis un rango exclusivo.

navegación es posible llegar al gestor de búsquedas, tal interacción se muestra en el *wireframe* de la Figura 2.71.

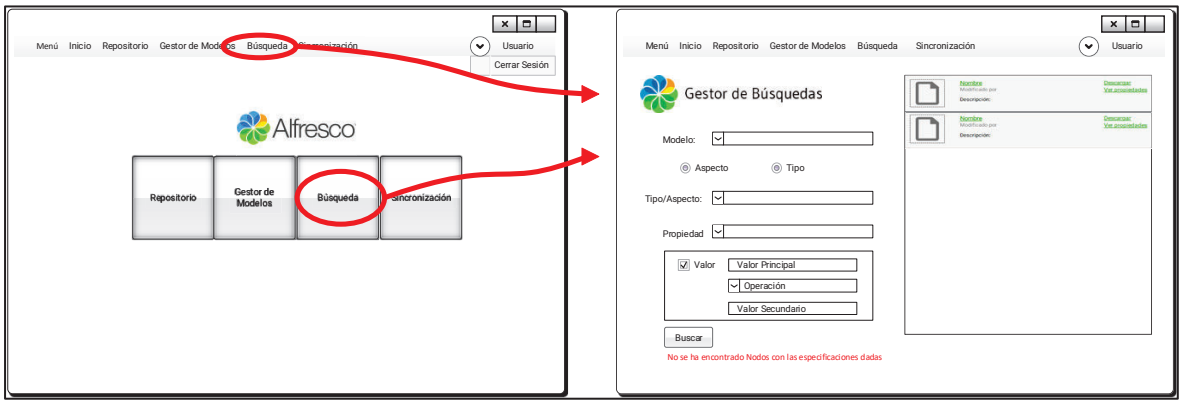


Figura 2.71 *Wireframe* de acceso al gestor de búsquedas

2.8.2 Clases e interfaces

Para el diseño de clases de este *sprint* se ha tomado como base la estructura Swagger proveniente de la Content Search API. Esta estructura provee una serie de clases que permiten crear búsquedas personalizadas.

En la Figura 2.73, se muestra la interfaz y las clases diseñadas para cumplir las funcionalidades de búsqueda. Se puede notar que esta interfaz cuenta con un solo *end-point*, sin embargo, la capacidad para hacer consultas complejas se encuentra en la estructura de clases, es por ello que la clase `BusquedaStatic` cuenta con la lógica para hacer consultas con objetos C#.

La estructura de clases se genera a partir del archivo Swagger de la Content Search API. El potencial de búsqueda que provee dicha estructura permite ejecutar consultas muy complejas [38]. Sin embargo, todo ese potencial puede ser resumido en consultas al estilo SQL utilizando el lenguaje de consulta de CMIS [39], basado en el estándar CMIS. A continuación, en la Tabla 2.15 se muestra algunas de las consultas utilizadas para la búsqueda de nodos. El lenguaje de consulta es especificado en la solicitud HTTP a través del objeto `RequestQuery` (ver Figura 2.72).

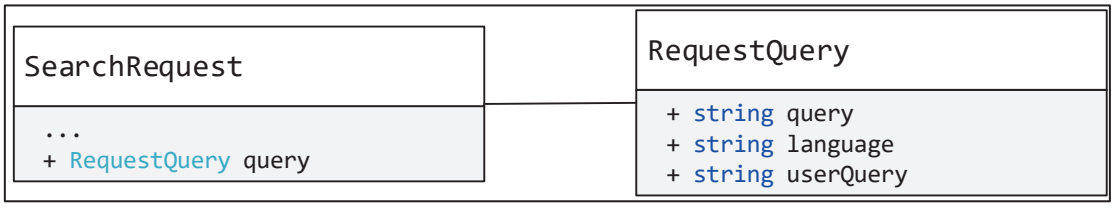


Figura 2.72 Diagrama de UML de clases para búsqueda de metadatos

Tabla 2.15 Algunas consultas utilizadas para la búsqueda de Nodos

Lenguaje	Consulta	Resultado
AFTS[52]	`ASPECT:"aspect.prefixedName"`	Todos los nodos que tengan aplicado el aspecto
AFTS	`TYPE:"type.prefixedName"`	Todos los nodos que sean de cierto tipo
AFTS	`property.prefixedName:*`	Todos los nodos que tengan cierta propiedad
AFTS	`property.prefixedName:"valor de la propiedad"`	Todos los Nodos con coincidencia exacta del valor de una propiedad
CMIS	`select * from type.prefixedName where propiedad.prefixedName > 'valorPropiedad'`	Todos los Nodos con valores de propiedad mayores que el valor especificado.

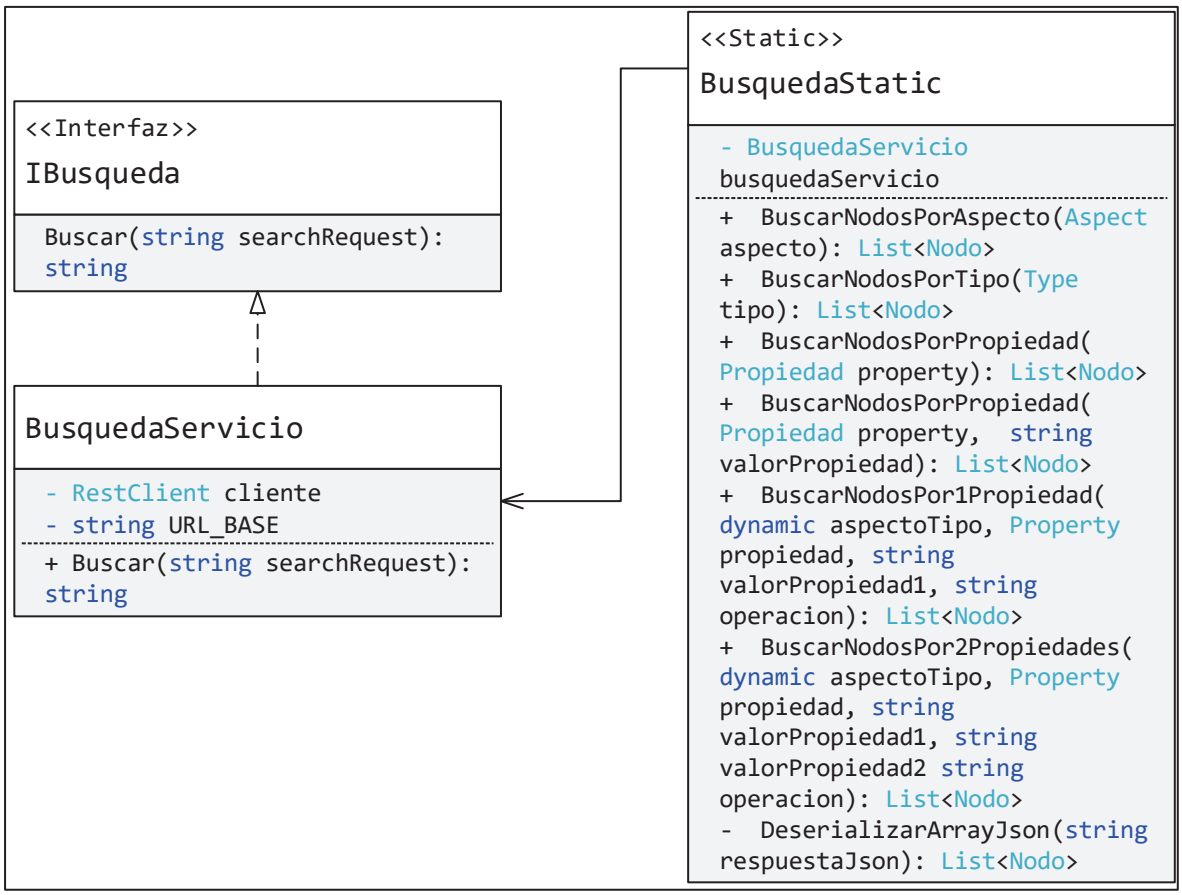


Figura 2.73 Diagrama de clases de comunicación para búsqueda de Nodos

En la Figura 2.72, se muestra el diagrama de clases utilizado para este *sprint*. En tal diagrama se puede observar que la clase `SearchRequest` contiene el atributo

[52] AFTS (*Alfresco Full Text Search*): Lenguaje de consulta propietario de Alfresco. Se ejecuta directamente contra la base de datos.

`RequestQuery`. La clase `RequestQuery` es la que contiene la cadena de consulta `query` que depende del lenguaje a utilizarse (`AFTS`, `LUCENE`[53], `CMIS`) y adicionalmente el atributo `userQuery` que se utiliza para enviar la consulta exacta del usuario.

2.8.3 Desarrollo

Para la interfaz gráfica de este *sprint* se han considerado los controles especificados en la Tabla 2.16.

Además, se codificaron interfaces y clases, necesarias para la comunicación cliente-servidor. En el Código 2.22 se muestra el método `BuscarNodosPor1Propiedada` que utiliza el lenguaje de consulta `CMIS` para que la consulta sea al estilo SQL; además este método dispone del parámetro de entrada `operacion` el cual condiciona la consulta que se realizará; en las consultas realizadas se compara el valor de la Propiedad del Nodo a buscar y el valor especificado como parámetro de entrada `valorPropiedad1`.

```
64          public static async Task<List<Nodo>> BuscarNodosPor1Propiedad(dynamic aspectoTipo,
65              Property propiedad, string valorPropiedad1, string operacion)
66          {
67              string query = "";
68              switch (operacion)
69              {
70                  case "Menor que":
71                      query = "select * from " + aspectoTipo.PrefixedName + " where " +
72                          propiedad.PrefixedName + "<'" + valorPropiedad1 + "'";
73                      break;
74                  case "Mayor que":
75                      query = "select * from " + aspectoTipo.PrefixedName + " where " +
76                          propiedad.PrefixedName + ">'" + valorPropiedad1 + "'";
77                      break;
78                  case "Mayor o igual que":
79                      query = "select * from " + aspectoTipo.PrefixedName + " where " +
80                          propiedad.PrefixedName + ">='" + valorPropiedad1 + "'";
81                      break;
82                  case "Menor o igual que":
83                      query = "select * from " + aspectoTipo.PrefixedName + " where " +
84                          propiedad.PrefixedName + "<='" + valorPropiedad1 + "'";
85                      break;
86                  default:
87                      break;
88              }
89              RequestQuery requestQuery = new RequestQuery(query);
90              requestQuery.Language = "cmis";
91              SearchRequest searchRequest = new SearchRequest();
92              searchRequest.Query = requestQuery;
93              string searchRequestJson = JsonConvert.SerializeObject(searchRequest);
94              string respuestaJson = await busquedaServicio.Buscar(searchRequestJson);
95              return DeserializarArrayJson(respuestaJson);
96          }
```

Código 2.22 Método `BuscarNodosPor1Propiedad`

[53] Lucene: Lenguaje de consulta compatible con Apache Lucene (*software* para búsqueda)

Tabla 2.16 Controles accionables para búsqueda de contenido

Nombre	Control	Acción
cmbxModelo	ComboBox	Permite seleccionar un Modelo
radioBtnTipo	RadioButton	Permite seleccionar la búsqueda de Tipos del modelo seleccionado
radioBtnAspecto	RadioButton	Permite seleccionar la búsqueda de Aspectos del modelo seleccionado
cmbxTipoAspecto	ComboBox	Permite seleccionar un Tipo o un Aspecto, depende de los RadioButton
cmbxPropiedad	ComboBox	Permite seleccionar una Propiedad perteneciente al Tipo o Aspecto seleccionado
chkboxValor	CheckBox	Permite activar los controles para especificar un valor(es) de la Propiedad seleccionada
txtValor	TextBox	Permite especificar un valor tipo string, int o float como parámetro de búsqueda.
txtValor2	TextBox	Permite especificar un valor int o float como segundo parámetro de búsqueda.
chkboxValor	ComboBox	Permite especificar un valor bool como parámetro de búsqueda.
dtpkrValor	DateTimePicker	Permite especificar un valor DateTime como parámetro de búsqueda.
dtpkrValor2	DateTimePicker	Permite especificar un valor DateTime como segundo parámetro de búsqueda.
cmbxOperacion	ComboBox	Permite seleccionar la operación de comparación a efectuarse, ya sea con uno o con dos parámetros de búsqueda.
flwlypanelNodos	FlowLayoutPanel	Permite desplegar los nodos encontrados

2.8.4 Entregables

El entregable de este *sprint* se muestra en la Figura 2.74.

2.8.5 Revisión y Retrospectiva

1. Se cambió el lugar de los filtros. La idea es que se puede apreciar de manera correcta los nodos encontrados.
2. Se corrigió ciertos errores al momento de formatear las fechas para la búsqueda, el formato para las fechas es el 8601 (2004).
3. Se agregó títulos a la interfaz gráfica debido a que esta carecía de los mismos y generaba confusión.
4. Se agregó iconos a los nodos encontrados, dependiendo de su extensión.

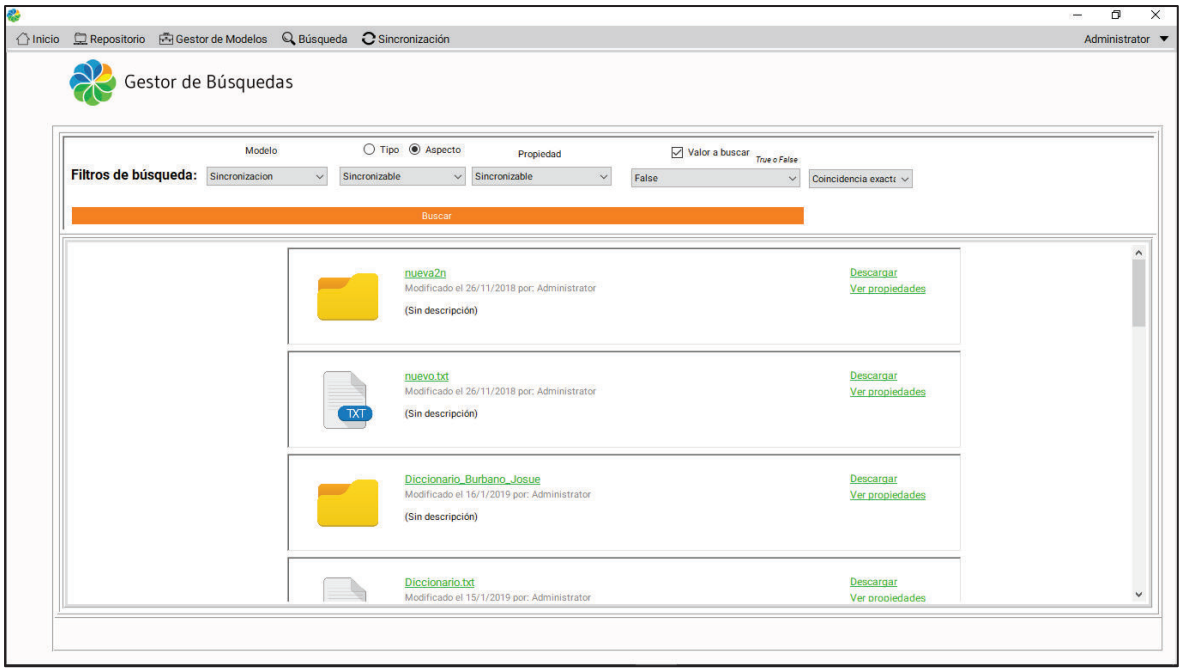


Figura 2.74 Pantalla del gestor de búsquedas del entregable del *sprint* 6

2.9 Diagrama UML de clases e interfaces

El diagrama UML de toda la aplicación se muestra en el ANEXO III.

3 RESULTADOS Y DISCUSIÓN

En esta sección se presentan los resultados de las pruebas de integración del prototipo, las pruebas de validación con el usuario y la corrección de los errores detectados en las pruebas.

Para las pruebas de integración se usó el cliente por defecto de Alfresco que permite verificar algunos de los cambios en los contenidos realizados desde el prototipo. Además, se realizaron solicitudes HTTP `GET` que permitan validar cambios no reflejados en el cliente por defecto de Alfresco. En este apartado se validará los requerimientos sistemáticos del prototipo.

Las pruebas de validación con el usuario se realizaron a través de una encuesta realizada a veinte docentes quienes a través de una serie de preguntas permitieron validar el funcionamiento, desde la perspectiva del usuario, del prototipo.

En esta sección también se muestran los errores detectados en las pruebas de integración y en las pruebas de validación de usuario.

3.1 Pruebas de integración

Las pruebas de integración se las realizó basándose en los requerimientos planteados en el levantamiento de la información. Algunas de las pruebas fueron usando el cliente por defecto de Alfresco y otras pruebas a través de solicitudes HTTP `GET`. Cuando el cliente por defecto de Alfresco no permita ver los cambios realizados en el servidor en el caso de que su interfaz gráfica no lo permita, entonces se recurre a realizar solicitudes `GET` a través de Postman, Javascript o desde un pequeño programa realizado en C#.

Las pruebas realizadas se muestran por requerimiento:

3.1.1 Iniciar sesión con las credenciales de Alfresco

En la aplicación se realiza un inicio de sesión con las credenciales de administrador y a través de Wireshark se captura algunas solicitudes y respuestas para encontrar el *ticket* de autenticación. Mediante este *ticket* se realiza una solicitud `GET` desde Postman, utilizando la Authorization API y se comprueba mediante el *ticket* de administrador que se ha iniciado sesión.

En la Figura 3.1 se muestra el inicio de sesión con las credenciales de administrador. En la Figura 3.2 se muestra que el usuario administrador ha ingresado al sistema debido a que su nombre aparece en la parte superior de la pantalla del prototipo.

Figura 3.1 Inicio de sesión con las credenciales de administrador

Figura 3.2 El usuario Administrator luego de iniciar sesión

Si el usuario ingresa con credenciales erróneas, entonces la pantalla vuelve a su estado inicial y se muestra un mensaje de error en letras rojas, como se muestra en la Figura 3.3.

Figura 3.3 Pantalla de inicio de sesión luego de un inicio de sesión erróneo

3.1.2 Cerrar sesión

Para cerrar sesión se utiliza el botón desplegable, donde se muestra el nombre de usuario, (ver Figura 3.2). Este botón permite eliminar el *ticket* de autenticación. Por otra parte, cuando se realiza la solicitud `GET` de la Authorization API que permite verificar si un *ticket* aún es válido y por consiguiente conocer si el usuario aún tiene una sesión activa en Alfresco, se puede observar en la Figura 3.4 (línea resaltada) que la respuesta indica que la autenticación ha fallado.

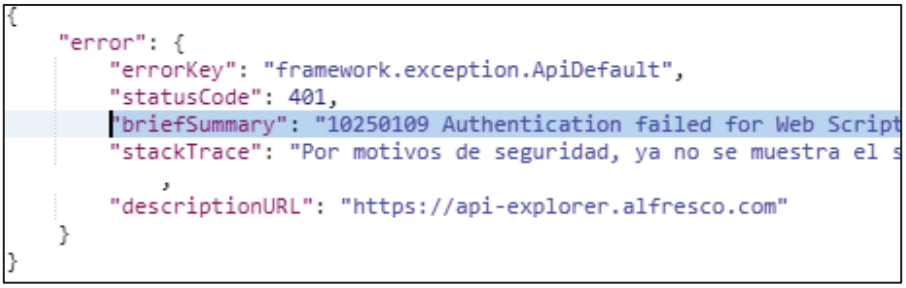


Figura 3.4 Respuesta a la solicitud de verificación de autenticación

Cuando un usuario cierra sesión a parte de eliminar su *ticket* de autenticación, la aplicación mostrará el mensaje (ver Figura 3.5). Luego de hacer clic en Aceptar se volverá a la pantalla de inicio de sesión (ver Figura 3.9).

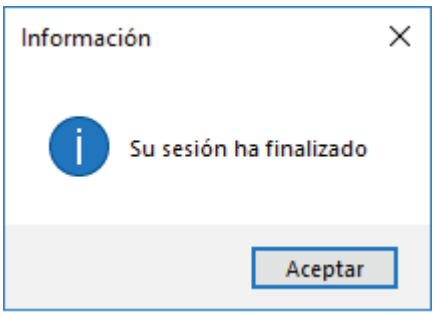


Figura 3.5 Aviso de finalización de sesión

3.1.3 Ingresar a un formulario de acceso a los demás módulos

Una vez que el usuario ha iniciado sesión en la aplicación se presenta el formulario mostrado en la Figura 3.6. Este formulario permite acceder a los diferentes módulos de la aplicación tanto desde sus botones de navegación ubicados en la parte central, como desde el menú de usuario ubicado en la parte superior. En la Figura 3.7 se muestra el formulario que se ha abierto luego de estar en el módulo Navegador de Repositorio y hacer clic sobre la opción del menú de usuario Sincronización.

Figura 3.6 Formulario de acceso a los diferentes módulos

Figura 3.7 Formulario para sincronización, accedido a través del menú de usuario

3.1.4 Permitir la visualización de archivos y carpetas de usuario

El Navegador de Repositorio es el encargado de realizar la visualización y navegación de archivos y carpetas de usuario (ver Figura 3.8). Este módulo permite navegar entre archivos y carpetas a través del control `TreeView` ubicado en la parte derecha o a través del nombre de la carpeta ubicada en el panel derecho. De hacer clic en el nombre de uno

de los elementos del control `TreeView`, entonces se mostrará el contenido de esta carpeta (Figura 3.9).

Para la validación de este requerimiento se ha verificado que los archivos presentes en Alfresco Share coincidan con los archivos presentes en la aplicación. Para ello se muestra el contenido de la carpeta `Adjuntos IMAP2` en el prototipo (Figura 3.9) y la misma carpeta vista desde Alfresco Share (Figura 3.10). Se puede apreciar que coinciden.

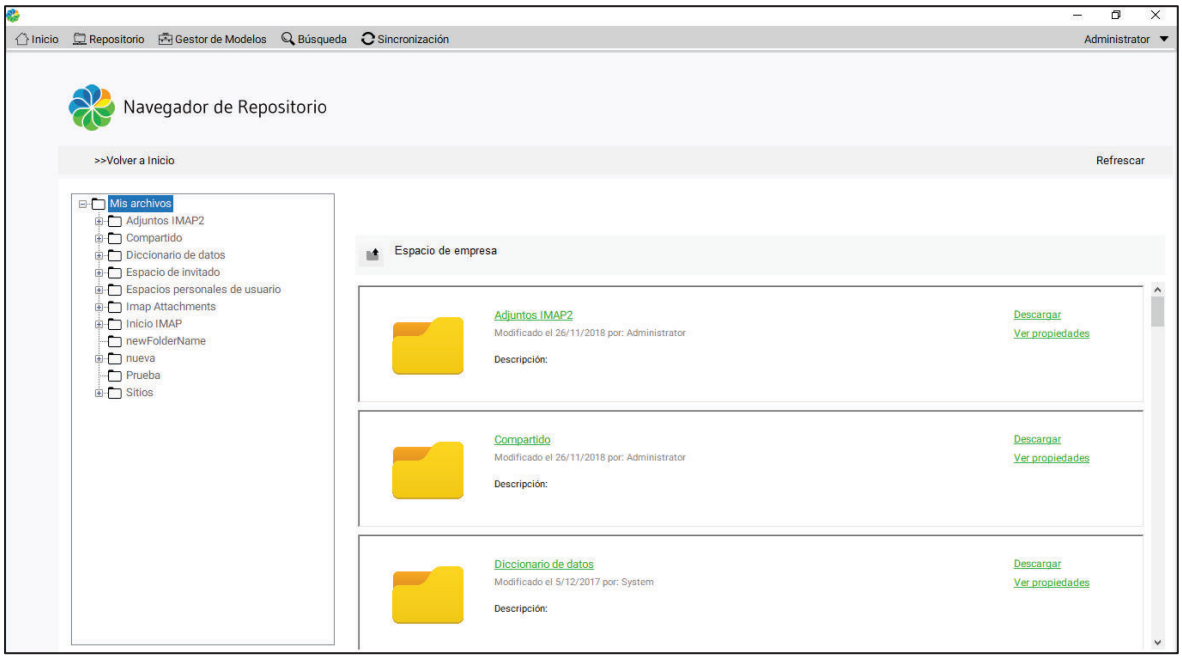


Figura 3.8 Módulo Navegador de Repositorio

Figura 3.9 Carpeta `Adjuntos IMAP2` vista desde el prototipo

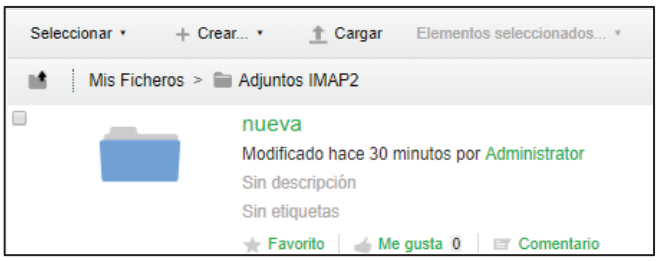


Figura 3.10 Carpeta `Adjuntos IMAP2` vista desde Alfresco Share.

3.1.5 Cargar masivamente contenidos a través de arrastrar y soltar

Para validar esta funcionalidad se realiza una carga que contenga ocho niveles de carpetas con su respectivo contenido cada una con archivos dentro arrastrando desde el explorador de archivos y soltado en la aplicación. La carpeta tiene de nombre `Explorador` y dentro de esta se encuentran la carpeta `Nivel 1` y el archivo `Archivo nivel 1.txt`. A su vez la carpeta `Nivel 1` contiene una carpeta y un archivo y así sucesivamente hasta completar los ocho niveles. Cuando se arrastra la carpeta `Explorador` al panel derecho del navegador de Repositorio (Figura 3.6) lo que sucede es que se cargan los archivos remotamente, se refresca el repositorio y se muestra un mensaje de confirmación. En la Figura 3.11 se muestra, en Alfresco Share, el último nivel cargado y en la Figura 3.12 se muestra el mensaje de confirmación.

Figura 3.11 Constancia de carpeta cargada a través de arrastrar y soltar

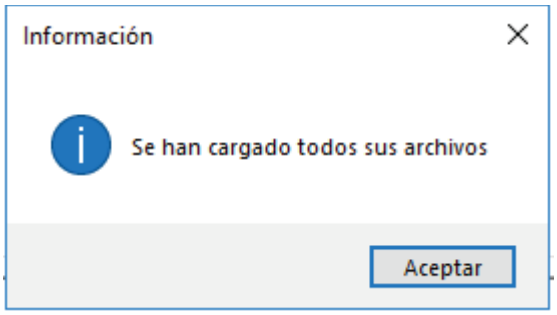


Figura 3.12 Confirmación de haber cargado múltiples archivos y carpetas

3.1.6 Permitir leer y editar metadatos de archivos y carpetas

Desde el Navegador de Repositorio (ver Figura 3.8) haciendo clic sobre la opción `Propiedades` presente en cada archivo o carpeta desplegada, se presentará el formulario de la Figura 3.13. Este formulario contiene algunos metadatos del archivo (nombre, versión,

última fecha de modificación, último modificador, tipo, título) y metadatos personalizados como Aspectos y sus Propiedades, y las Propiedades del Tipo.

El contenido de la Figura 3.15 puede ser cambiado de Tipo y muchos Aspectos pueden ser agregados, de ello dependerá el número de Propiedades que se muestren. En la Figura 3.15 se muestran Nombre, Tipo y *Title* debido a que el Tipo es el Tipo por defecto (`cm:content`) y no tiene ningún aspecto agregado.

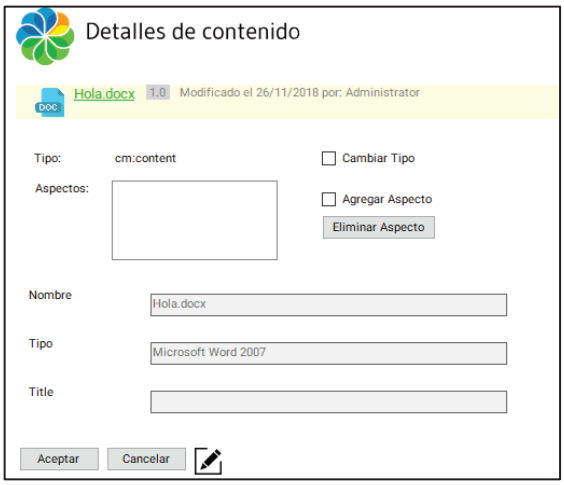


Figura 3.13 Metadatos del archivo `Hola.docx`

El archivo `Hola.docx` fue cambiado al Tipo `alfepn:Prueba` y se le agrego el Aspecto `myAspect1`, como se muestra en la Figura 3.14. `Hola.docx` ahora cuenta con las Propiedades provenientes tanto del nuevo Tipo (`alfepn:Prueba`) como del Aspecto que se ha agregado (`myAspect1`).

Figura 3.14 Metadatos personalizados del archivo `Hola.docx`

Para agregar el valor a una Propiedad, se debe presionar el botón de edición (que tiene un lápiz como icono), se edita la Propiedad y luego de hacer clic en el botón `Aceptar` se actualizan las propiedades y se muestra un mensaje de confirmación. Se actualizó la propiedad `mostrable_etq` con el valor "`Modificado para pruebas`", en este caso las propiedades actualizadas se muestran en la Figura 3.15, mientras que el mensaje de confirmación en la Figura 3.16.

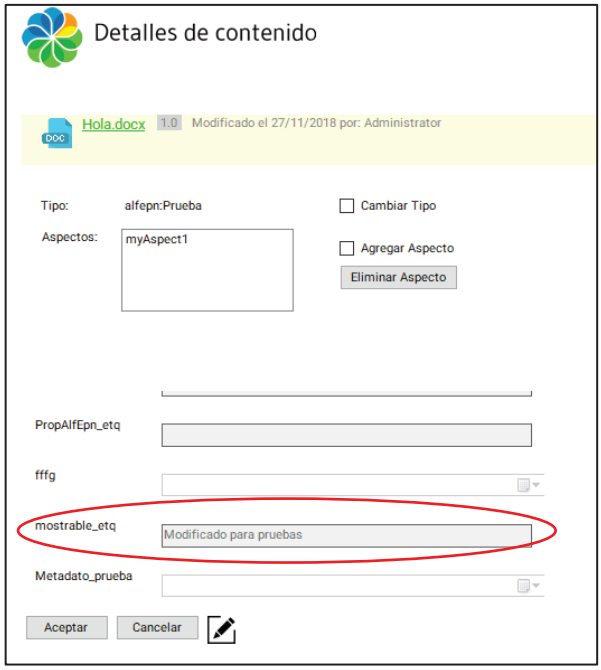


Figura 3.15 Propiedades actualizadas

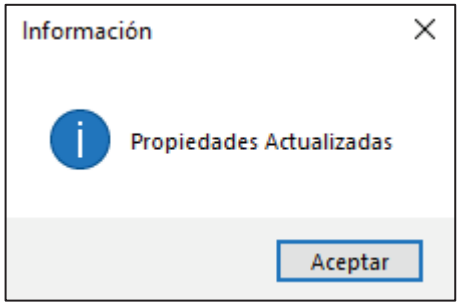


Figura 3.16 Mensaje de confirmación para edición de propiedades

3.1.7 CRUD de Modelos de Contenido

Para validar este requerimiento se ha creado un modelo de contenido desde el prototipo y se ha verificado cambios en Alfresco Share. Desde el formulario de la Figura 3.17 se ha creado, se ha editado y se ha eliminado un Modelo. A lo largo de cada acción se ha ido verificando mensajes de advertencia y error que posiblemente surgirán al interactuar con

este modelo. Para crear, editar o eliminar un Modelo se cuenta con un menú contextual, el cual se muestra en la Figura 3.18.

Se ha creado el Modelo `Modelo_prueba` con los atributos mostrados en la Figura 3.20. El mensaje de confirmación de creación se muestra en la Figura 3.19.

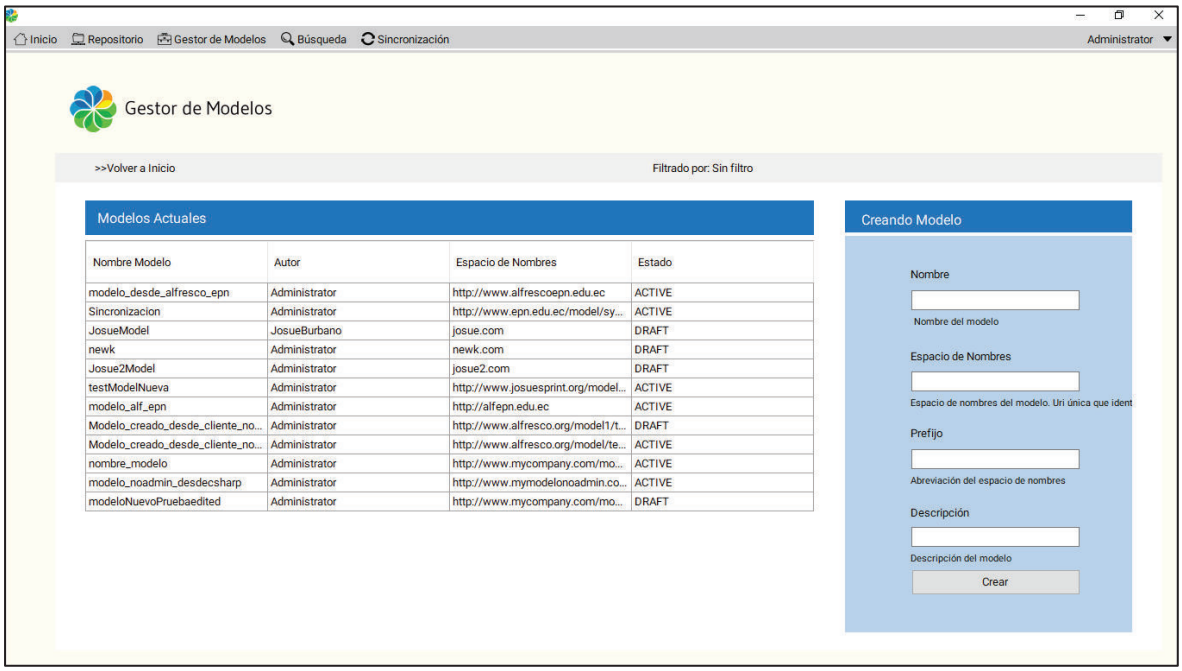


Figura 3.17 Gestor de Modelos

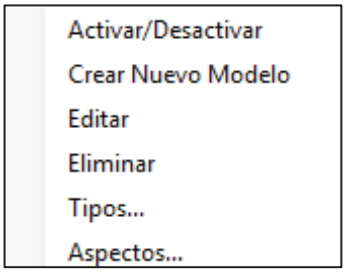


Figura 3.18 Menú contextual para crear, editar y eliminar Modelos

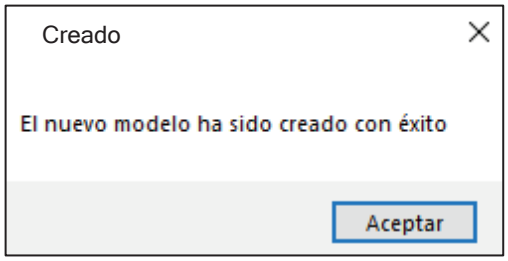


Figura 3.19 Mensaje de confirmación de creación de un modelo

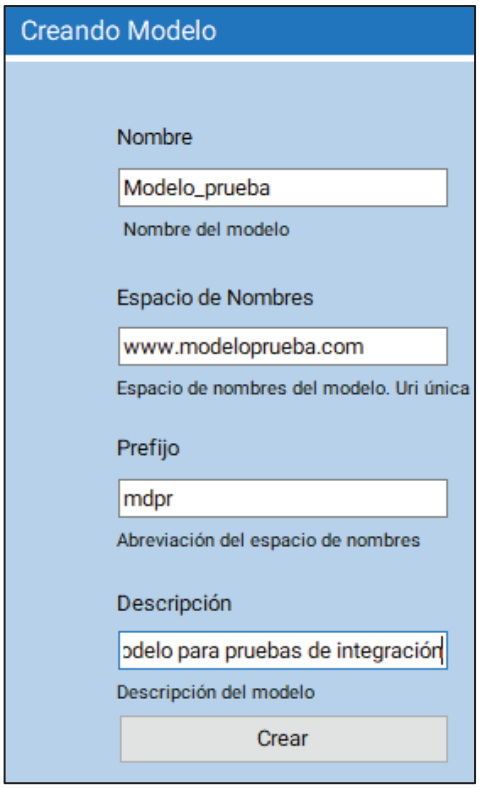


Figura 3.20 Creación de un modelo de prueba

De igual manera, cuando se edita un Modelo, los cambios se ven reflejados en Alfresco Share. En la Figura 3.21 se ha modificado el modelo `Modelo_prueba`, se ha cambiado el espacio de nombres de `www.modeloprueba.com` a `www.prueba.com` y el prefijo de `mdpr` a `mdpr2`. El mensaje de confirmación, luego de accionar el botón `Editar` se muestra en la Figura 3.22.

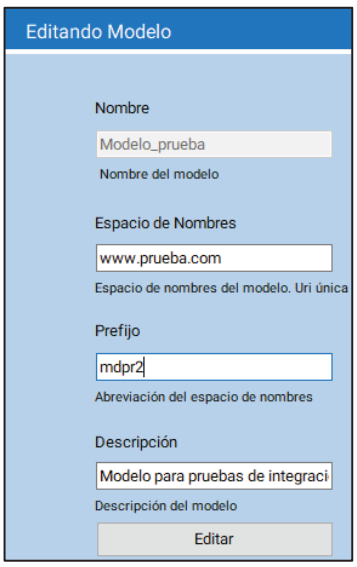


Figura 3.21 Editando el Modelo `Modelo_prueba`

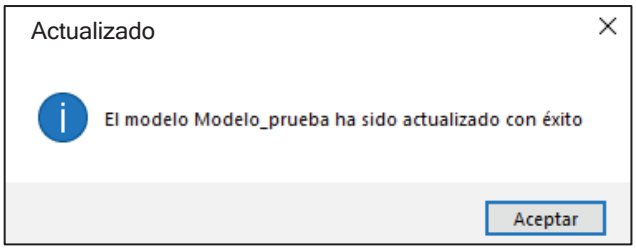


Figura 3.22 Mensaje de confirmación de edición exitosa

Cuando un modelo se elimina se muestra un mensaje de confirmación indicando que efectivamente el modelo se ha eliminado en el servidor. Para constatar esto, se hace una solicitud `GET` desde Postman pidiendo el Modelo de nombre `Modelo_prueba`, como este modelo ha sido eliminado entonces la respuesta indica que no se ha encontrado el modelo, como se muestra en la línea sombreada de la respuesta de la Figura 3.24.

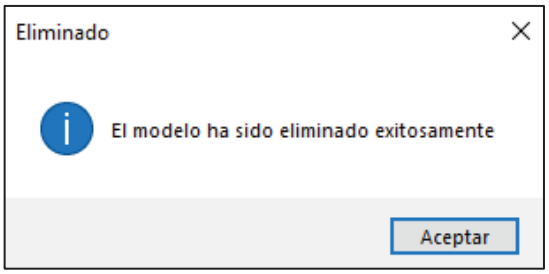


Figura 3.23 Mensaje de confirmación de eliminación del `Modelo_prueba`

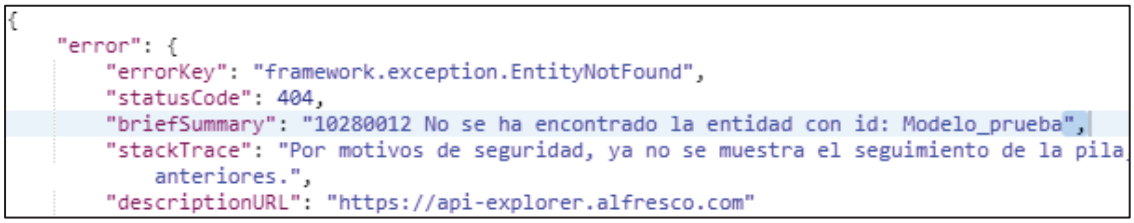


Figura 3.24 Respuesta al `GET` solicitando el modelo de `Modelo_prueba`

Por otra parte, si se intenta eliminar un Modelo activo, entonces se mostrará el mensaje de error de la Figura 3.25.

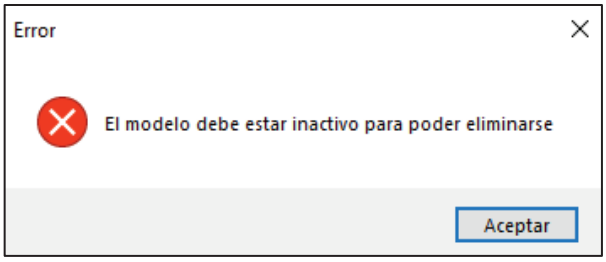


Figura 3.25 Mensaje de error al intentar eliminar un Modelo activo

Este formulario cuenta con validaciones de tipeo, campos requeridos, nombres duplicados y permisos insuficientes. A continuación, se muestran los mensajes de advertencia o error mostrados para cada uno de los casos. Si se intenta ingresar caracteres no permitidos, el mensaje mostrado es el de la Figura 3.26.

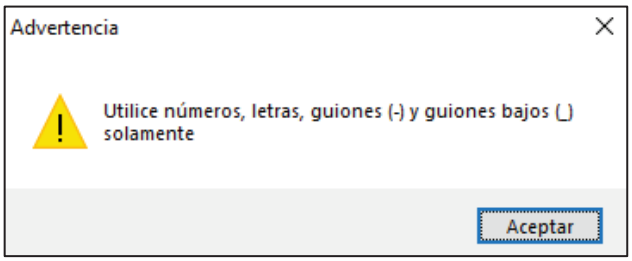


Figura 3.26 Mensaje de error cuando se intenta ingresar caracteres no permitidos

Si uno de los campos requeridos (`Nombre`, `Espacio de Nombres` o `Prefijo`) no se ha completado, y se acciona el botón `Crear` entonces se mostrará un mensaje indicando que estos campos son obligatorios, como se ve en la Figura 3.29.

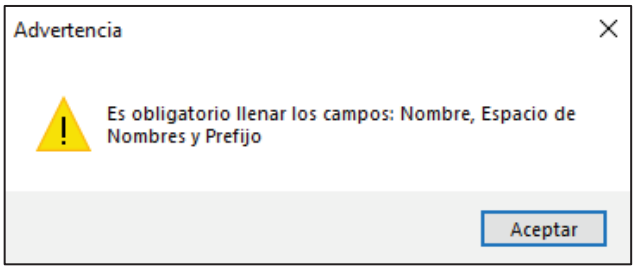


Figura 3.27 Mensaje de error cuando no se han llenado los campos obligatorios

Por otra parte, cuando el nombre del Modelo, el espacio de nombres o el prefijo ya han sido utilizados y se está intentado crear un Modelo con dicha información duplicada entonces el mensaje que se mostrará será el de la Figura 3.28.

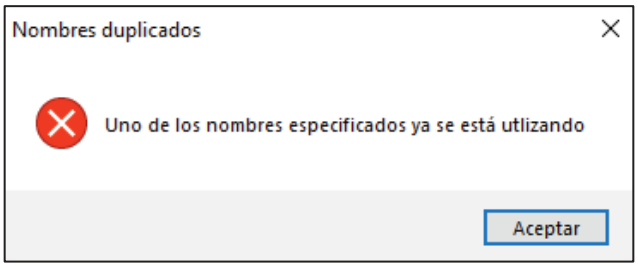


Figura 3.28 Mensaje de error cuando un nombre ya se está utilizando

Cuando un usuario que no tiene el permiso para crear, editar o eliminar un Modelo, intenta hacerlo, entonces se muestra el mensaje de la Figura 3.29 indicando que el usuario carece

de los permisos suficientes ya que no pertenece al grupo ALFRESCO_MODEL_ADMINISTRATORS.

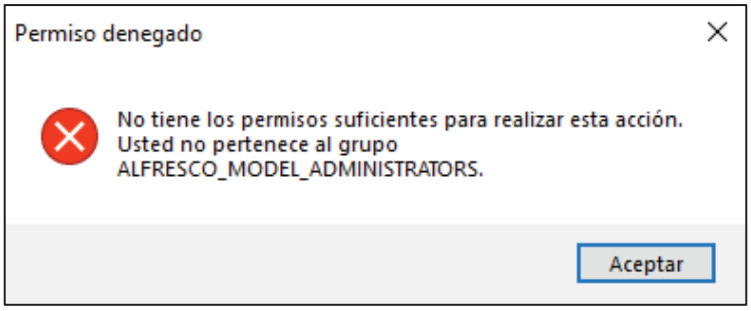


Figura 3.29 Mensaje de error al crear un Modelo sin el permiso necesario

3.1.8 Permitir la activación y desactivación de Modelos de Contenido

Para validar este requerimiento se tomó tres Modelos del gestor de modelos, un modelo activo y dos inactivos. A todos se les intentó cambiar de estado. Los tres modelos son `testModelNueva`, `modelo_alf_epn` y `Modelo_creado_desde`. En la Figura 3.30 se muestran tales modelos, en la primera columna está su nombre y en la última su estado. El modelo `testModelNueva` esta inicialmente activo, el modelo `model_alf_epn` está inicialmente inactivo y el modelo `Modelo_creado_desde` está activo, en este caso `DRAFT` significa inactivo.

testModelNueva	Administrator	http://www.josuesprint.org/model...	ACTIVE
modelo_alf_epn	Administrator	http://alfepn.edu.ec	DRAFT
Modelo_creado_desde_cliente_no...	Administrator	http://www.alfresco.org/model1/t...	ACTIVE

Figura 3.30 Modelo `testModelNueva` (Activo) y `modelo_alf_epn` (Inactivo)

Al modelo `testModelNueva` se lo intentó cambiar de activo a inactivo, sin embargo, este modelo está siendo usado en algún contenido y es por ello que no se permite la desactivación de este modelo. El mensaje de error de esta petición se muestra en la Figura 3.31.

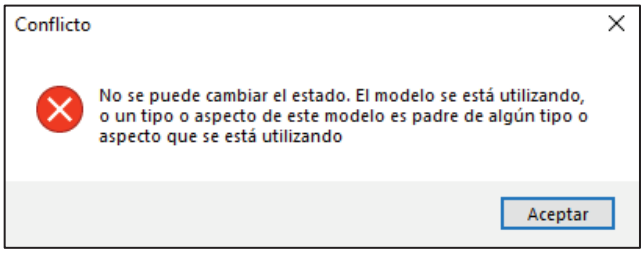


Figura 3.31 Mensaje de error al intentar desactivar un Modelo utilizado

Al modelo `modelo_alf` se lo ha cambiado de estado inactivo a estado activo. El mensaje de confirmación de cambio de estado se muestra en la Figura 3.32.

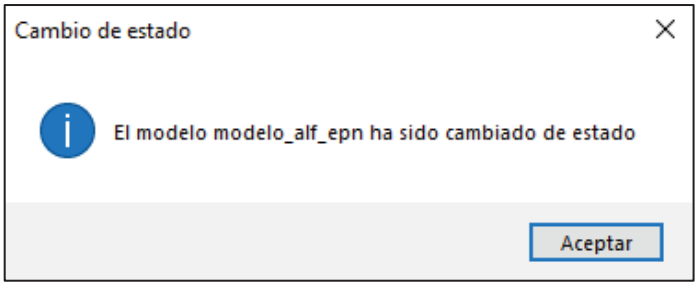


Figura 3.32 Mensaje de confirmación de cambio de estado

El Modelo `Modelo_creado_desde` se lo cambió de activo a inactivo. Como este modelo no está siendo utilizado sobre ningún contenido, entonces no existe ningún problema en cambiarlo de estado. El mensaje de confirmación que se mostrará es el mismo de la Figura 3.32.

3.1.9 CRUD de Tipos

Para validar este requerimiento se ha manipulado un Tipo desde el prototipo. Para ello desde el formulario de la Figura 3.33 y a través del menú contextual de la Figura 3.34 se ha realizado las operaciones de creación, edición y eliminación de Tipos.

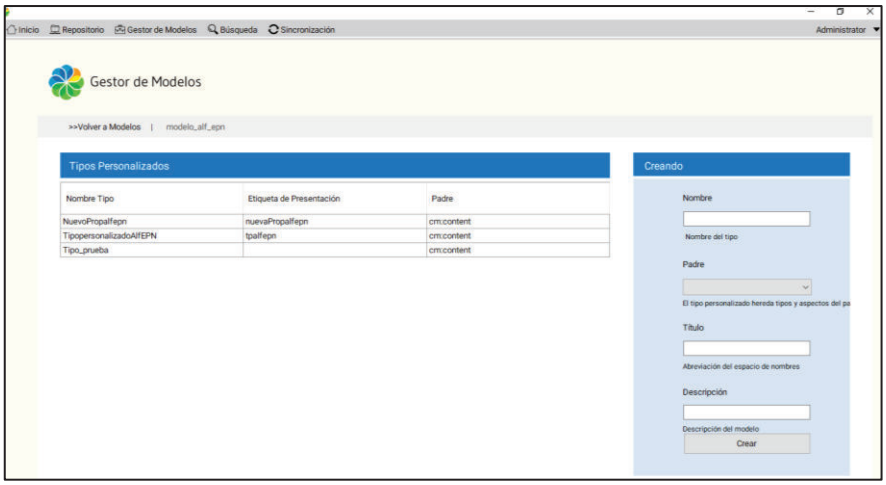


Figura 3.33 Gestor de Tipos

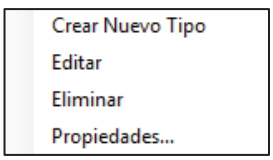


Figura 3.34 Menú contextual para crear, editar y eliminar Tipos

De igual manera, los mensajes de error y validaciones se irán mostrando conforme vayan sucediendo y al final de esta subsección.

Para crear un Tipo se cuenta con un formulario que permite ingresar sus diferentes parámetros, como se muestra en la Figura 3.35. El nombre del Tipo a crearse es `Tipo_prueba` y su padre es `cm:content`. Luego de presionar el botón `Crear` y se mostrará un mensaje de confirmación, informando que el Tipo se ha creado con éxito. El mensaje de confirmación se muestra en la Figura 3.36.

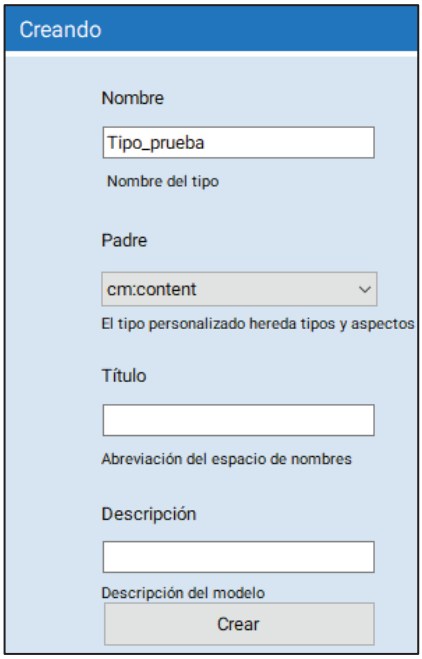


Figura 3.35 Creación de un Tipo de prueba

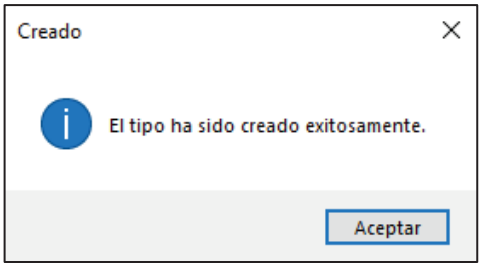


Figura 3.36 Mensaje de confirmación de creación del Tipo `Tipo_prueba`

Para validar la edición de Tipos, se intenta editar un Tipo de Modelo activo y un Tipo de Modelo inactivo. El Tipo de Modelo inactivo se lo editó satisfactoriamente, su edición se muestra en la Figura 3.37, ahí se puede observar que se agrega el título "`Título Tipo_prueba`" y se cambia el padre a `cm:folder`. El mensaje de confirmación luego de presionar el botón `Editar` se muestra en la Figura 3.38.

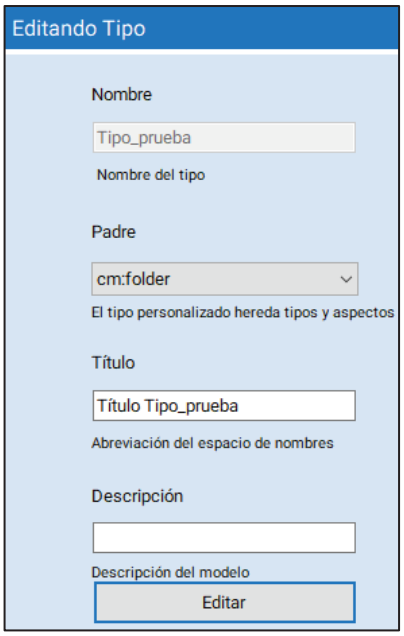


Figura 3.37 Editando Tipo `Tipo_prueba`

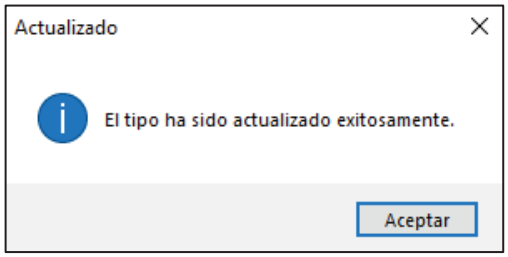


Figura 3.38 Mensaje de confirmación de la edición del Tipo `Tipo_prueba`

Cuando se intenta modificar el padre de un Tipo de Modelo activo que se está utilizando, entonces se muestra un mensaje de error como el de la Figura 3.39.

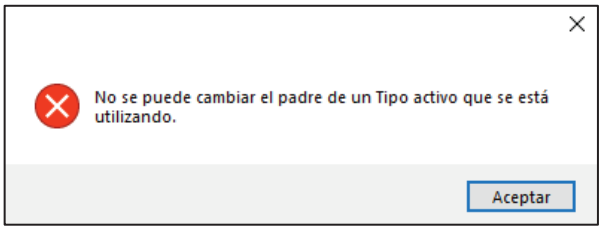


Figura 3.39 Mensaje de error al intentar editar el padre de un modelo activo

Cuando un Tipo ha sido eliminado, entonces todas sus Propiedades se eliminarán con él. El mensaje de confirmación de la eliminación de un Tipo se muestra en la Figura 3.40. Por otra parte, si el Tipo que se quiere eliminar pertenece a un Modelo activo entonces, la acción será rechazada y se mostrará el mensaje de la Figura 3.41.

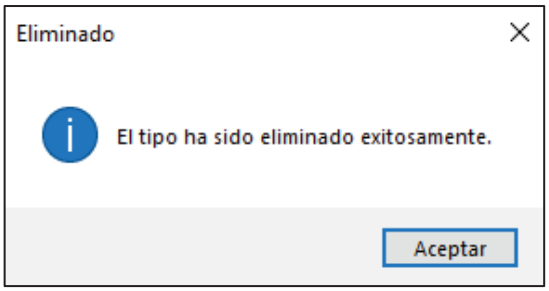


Figura 3.40 Mensaje de confirmación de la eliminación de un Tipo

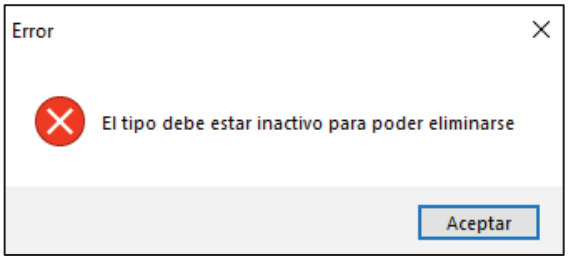


Figura 3.41 Mensaje de error al intentar eliminar un Tipo de Modelo activo

Para validar efectivamente que un Tipo ha sido eliminado se realiza una solicitud `GET` a través de Postman, pidiendo el Tipo `Tipo_prueba`. La respuesta indica que el Tipo con ese nombre no se encuentra como se aprecia en la línea resaltada de la Figura 3.42.


```
"error": {
    "errorKey": "framework.exception.EntityNotFound",
    "statusCode": 404,
    "briefSummary": "10290036 No se ha encontrado la entidad con id: Tipo_prueba",
    "stackTrace": "Por motivos de seguridad, ya no se muestra el seguimiento de la pila,
        anteriores.",
    "descriptionURL": "https://api-explorer.alfresco.com"
```


Figura 3.42 Respuesta a la solicitud `GET` después de eliminar el Tipo `Tipo_prueba`

El formulario gestor de Tipos también cuenta con validaciones antes de que un Tipo sea creado. El campo `Nombre` cuenta con validaciones para evitar el ingreso de caracteres no permitidos en el nombre de un Tipo. Si se ingresan caracteres no permitidos entonces se mostrará un mensaje como el de la Figura 3.43.

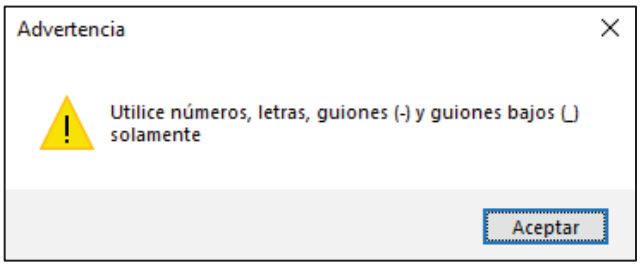


Figura 3.43 Mensaje de error cuando se intenta ingresar caracteres no permitidos

Cuando se intenta crear un tipo y no se ha seleccionado un padre o un nombre, entonces se muestra el mensaje de error de la Figura 3.44.

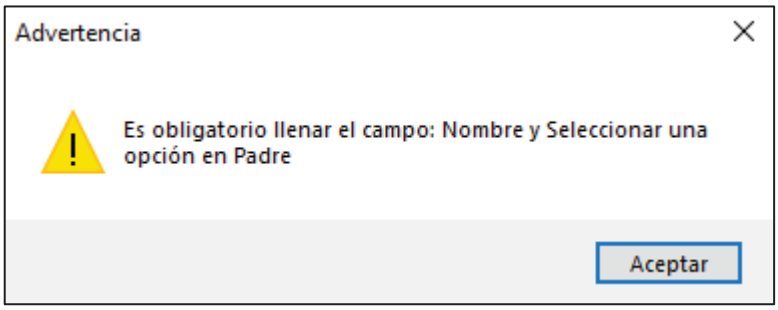


Figura 3.44 Mensaje de error derivado de no seleccionar un Padre para la creación

Cuando se intenta crear, editar o eliminar un Tipo sin tener los permisos necesarios, entonces la operación será abortada y el mensaje que se mostrará será el de la Figura 3.45.

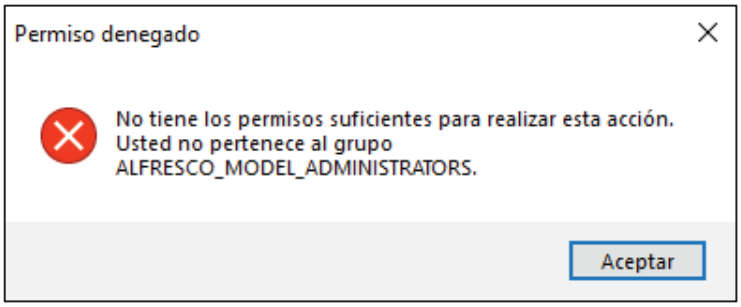


Figura 3.45 Mensaje de error cuando se el usuario no cuenta con permisos

3.1.10 CRUD de Aspectos

Para validar este requerimiento se ha manipulado un Aspecto desde el prototipo. Para ello desde el formulario de la Figura 3.47 y a través del menú contextual de la Figura 3.46 se ha realizado las operaciones de creación, edición y eliminación de Aspectos. Este formulario también cuenta con algunas validaciones y mensajes de error que se mostrarán conforme estos vayan sucediendo o en la parte final de esta subsección.

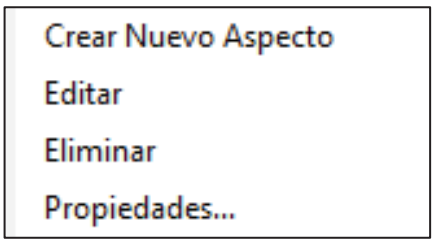


Figura 3.46 Menú contextual para crear, editar y eliminar Aspectos

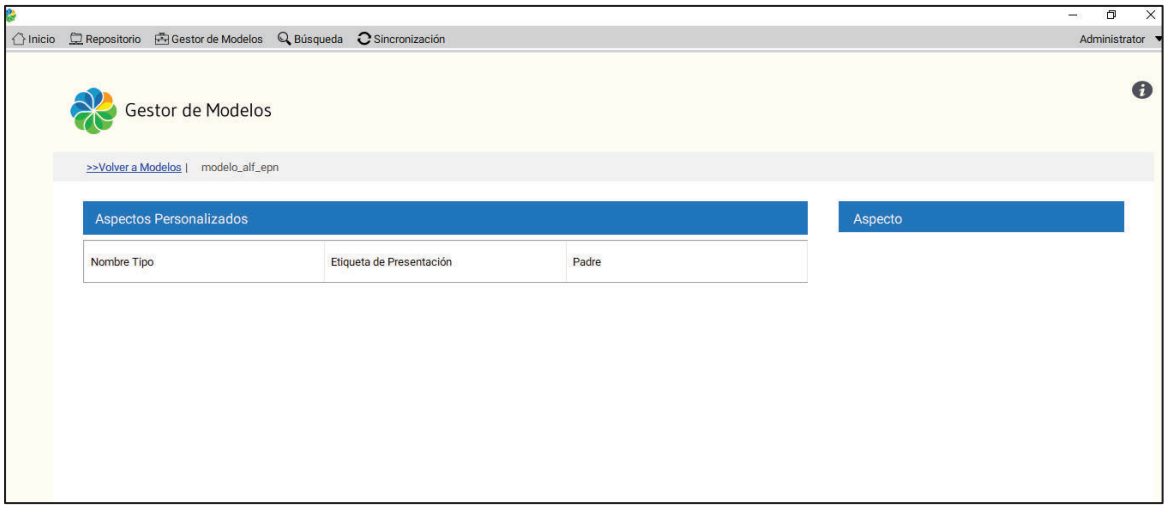


Figura 3.47 Gestor de Aspectos

Para crear un Aspecto se cuenta con un formulario que permite ingresar los diferentes parámetros de un Aspecto, como se muestra en la Figura 3.48. El nombre del Aspecto a crearse es `Aspecto_prueba`, los demás campos no son obligatorios. Luego de accionar el botón `Crear` y de estar todo en orden, se mostrará un mensaje de confirmación, informando que el Aspecto se ha creado con éxito.

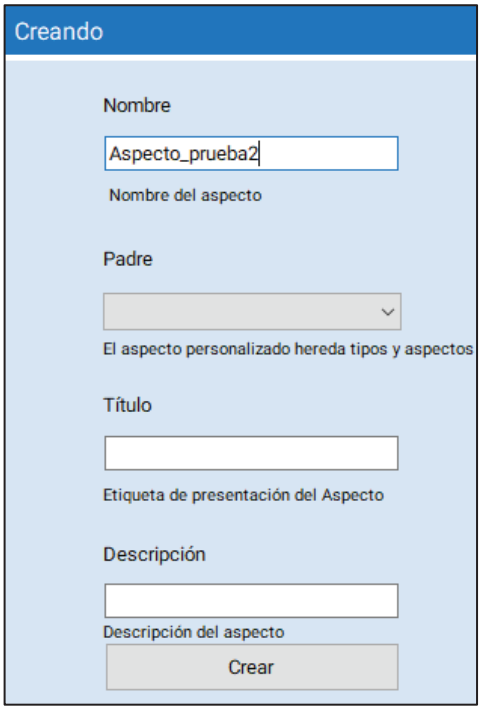


Figura 3.48 Creación de un Aspecto de prueba

El mensaje de confirmación de la acción anterior se muestra en la Figura 3.49.

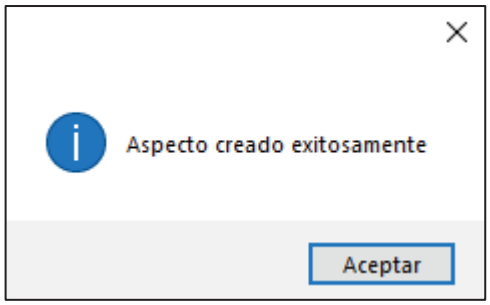


Figura 3.49 Mensaje de confirmación de creación del Aspecto `Aspecto_prueba2`

Para validar la edición de Aspectos, se intenta editar un Aspecto de Modelo activo y un Aspecto de Modelo inactivo. El Aspecto de Modelo inactivo se lo editó satisfactoriamente, su edición se muestra en la Figura 3.50, ahí se puede observar que se agrega la descripción "`Descripción de Aspecto prueba 2`" y se cambia el padre a `g2:editingIngoogle`. El mensaje de confirmación luego de accionar el botón `Editar` se muestra en la Figura 3.51.

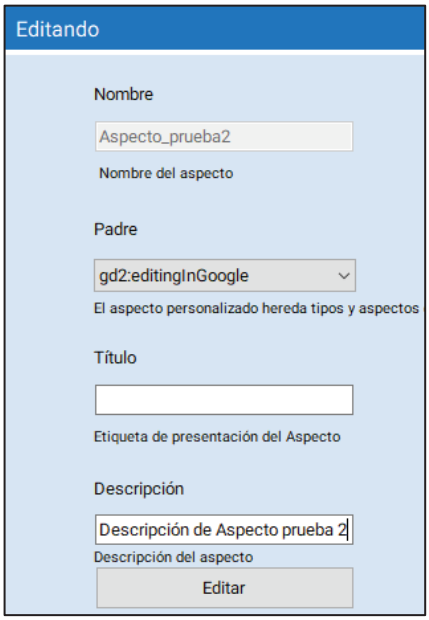


Figura 3.50 Editando Tipo `Tipo_prueba`

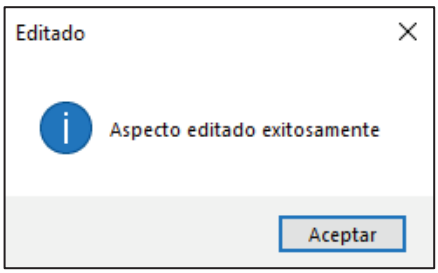


Figura 3.51 Mensaje de confirmación de la edición del Aspecto `Aspecto_prueba2`

Cuando se intenta modificar el padre de un Aspecto de Modelo activo que se está utilizando, entonces el mensaje de error que se muestra es el de la Figura 3.52.

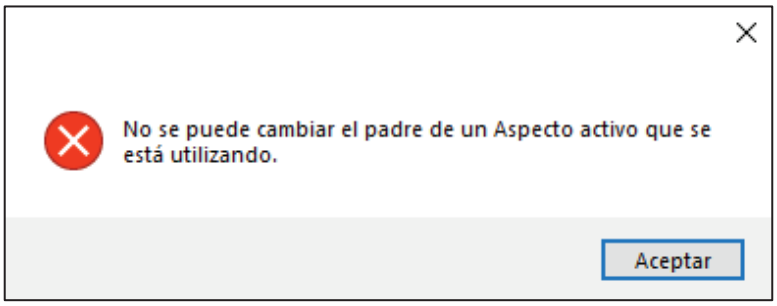


Figura 3.52 Mensaje de error al intentar editar el padre de un modelo activo

Cuando un Aspecto ha sido eliminado, entonces todas sus Propiedades se eliminarán con él. El mensaje de confirmación de la eliminación de un Aspecto se muestra en la Figura 3.53. Por otra parte, si el Aspecto que se quiere eliminar pertenece a un Modelo activo entonces, la acción será rechazada y se mostrará el mensaje de la Figura 3.54.

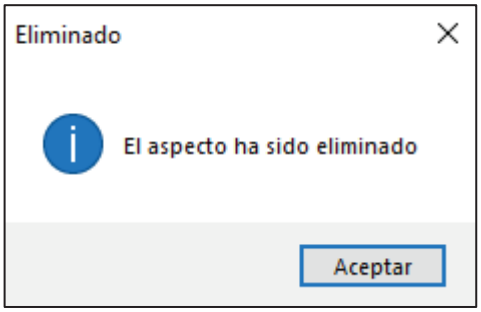


Figura 3.53 Mensaje de confirmación de la eliminación de un Aspecto

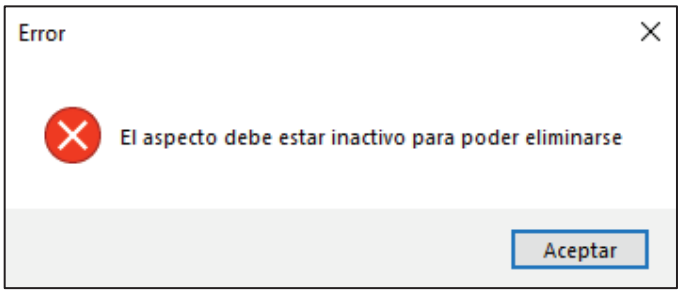


Figura 3.54 Mensaje de error al intentar eliminar un Aspecto de Modelo activo

Para validar efectivamente que un Tipo ha sido eliminado se realiza una solicitud `GET` a través de Postman, solicitando el Aspecto `Aspecto_prueba2`. La respuesta a esta solicitud es que el Aspecto con ese nombre no se encuentra como lo muestra la línea resaltada de la Figura 3.55

```
{
    "error": {
        "errorKey": "framework.exception.EntityNotFound",
        "statusCode": 404,
        "briefSummary": "10290044 No se ha encontrado la entidad con id: Aspecto_prueba2",
        "stackTrace": "Por motivos de seguridad, ya no se muestra el seguimiento de la pila,
            ,
        "descriptionURL": "https://api-explorer.alfresco.com"
    }
}
```


Figura 3.55 Respuesta a la solicitud `GET` después de eliminar el Tipo `Tipo_prueba`

El formulario gestor de Aspectos también cuenta con validaciones antes de que un Aspecto sea creado. El campo `Nombre` cuenta con validaciones para evitar el ingreso de caracteres no permitidos en el nombre de un Aspecto. Si se ingresan caracteres no permitidos entonces se mostrará un mensaje como el de la Figura 3.56.

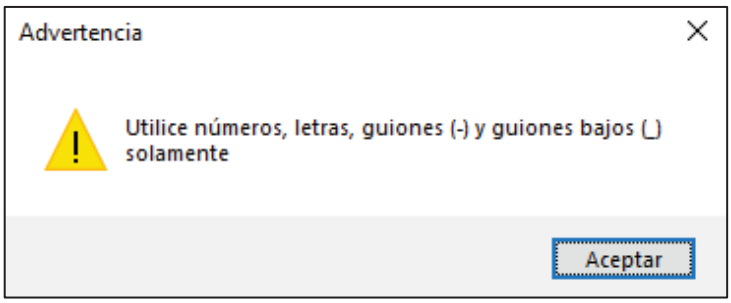


Figura 3.56 Mensaje de error cuando se intenta ingresar caracteres no permitidos

Cuando se intenta crear, editar o eliminar un Aspecto sin tener los permisos necesarios, entonces la operación será abortada y el mensaje que se mostrará será el de la Figura 3.57.

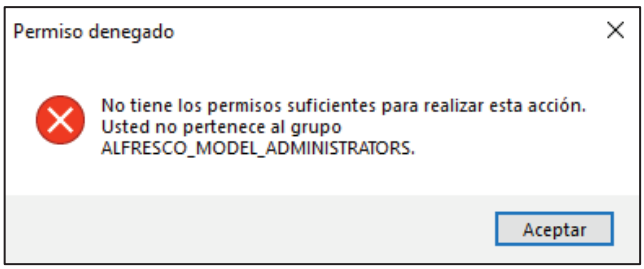


Figura 3.57 Mensaje de error cuando se el usuario no cuenta con permisos

3.1.11 CRUD de Propiedades

Para validar este requerimiento se manipuló una Propiedad desde el formulario gestor de Propiedades de la Figura 3.58. La acción de crear, editar o eliminar una Propiedad se la realiza a través del menú contextual de la Figura 3.59. A diferencia de los Tipos, Aspectos

y Modelos, una Propiedad es una estructura más compleja que requiere de un mayor detalle debido a que algunas de las combinaciones de sus campos no son compatibles, por ello en esta validación se muestran diferentes tablas con las combinaciones realizadas y probadas.

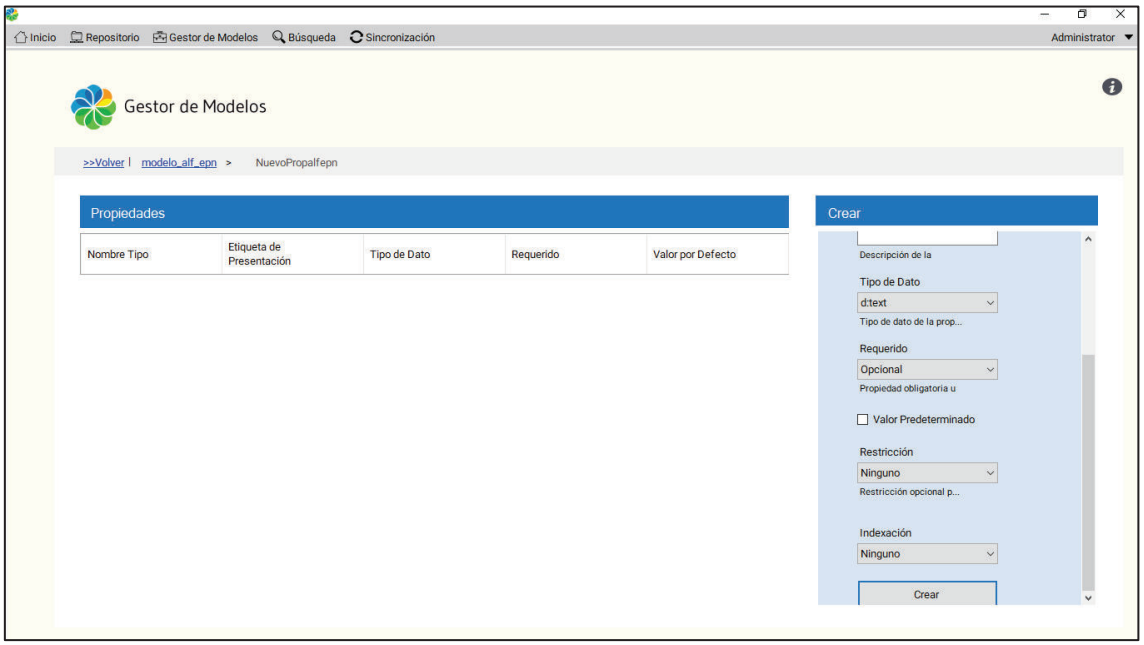


Figura 3.58 Gestor de Propiedades

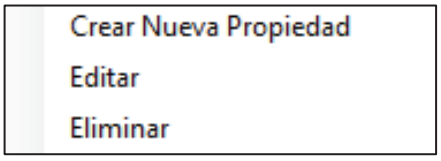


Figura 3.59 Menú contextual para creación, edición y eliminación de Propiedades

Para validar la creación de una nueva Propiedad se utilizó el panel de la Figura 3.61 el cual permite ingresar los campos que constituyen una Propiedad. El mensaje de confirmación de que la Propiedad `Propiedad_prueba` ha sido creada se muestra en la Figura 3.60.

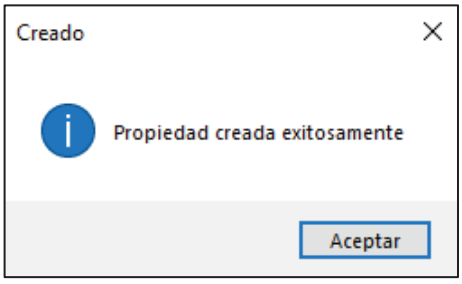


Figura 3.60 Confirmación de creación de la Propiedad `Propiedad_prueba`

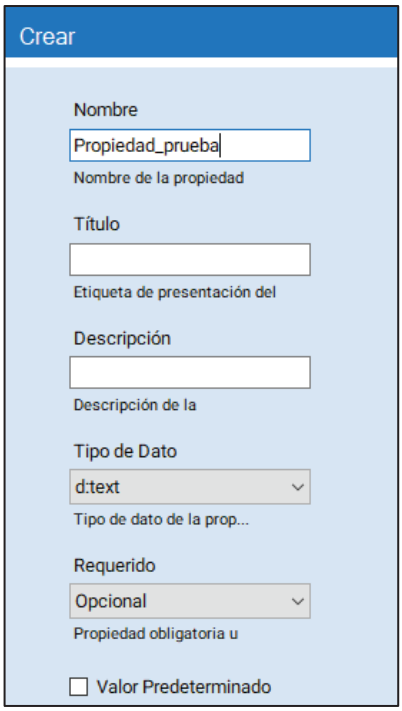


Figura 3.61 Creando la Propiedad `Propiedad_prueba`

Para validar la edición de una Propiedad se ha tomado la Propiedad `Propiedad_prueba` y se le ha añadido el valor "`Titulo propiedad`", como se muestra en la Figura 3.62. Luego de accionar el botón `Editar` la propiedad se actualizará y seguido a ello se mostrará el mensaje de la Figura 3.63.

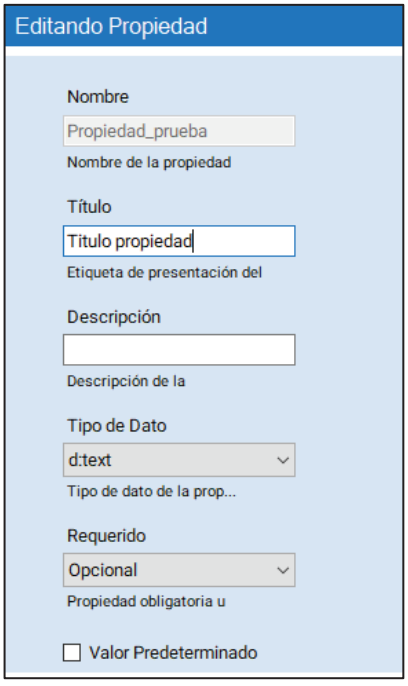


Figura 3.62 Editando la Propiedad `Propiedad_prueba`

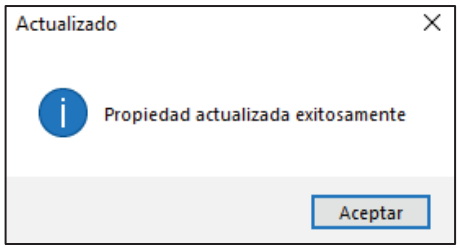


Figura 3.63 Mensaje de confirmación luego de editar una Propiedad

Luego de ello se ha eliminado la Propiedad `Propiedad_prueba`. El mensaje de confirmación de esta acción se muestra en la Figura 3.64. Posterior a ello y con el fin de validar de que efectivamente la Propiedad se ha eliminado entonces se realiza una solicitud `GET` solicitando la Propiedad que ha sido eliminada. La respuesta a esta solicitud se muestra en la Figura 3.65 y en la línea sombreada se puede observar que el Tipo sobre el que se creó la Propiedad `Propiedad_prueba` no contiene ninguna Propiedad.

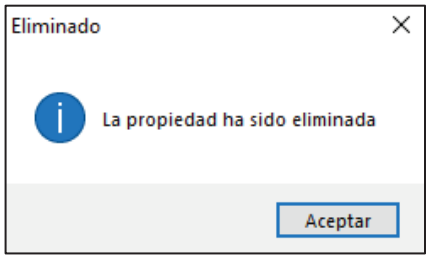


Figura 3.64 Mensaje de confirmación luego de eliminar una Propiedad

Figura 3.65 Respuesta a solicitud `GET`. Se muestra que no existen Propiedades

Para la validación de ingreso y posibles combinaciones de campos se ha tomado en cuenta la Tabla 3.1, la cual muestra el campo, los posibles valores a ingresarse, si existe algún tipo de restricción al momento de ingresar el valor de ese campo y si es que tal campo depende de otro campo. Por ejemplo, el campo `Requerido` tiene dos opciones definidas para ingresarse como posibles valores: `Opcional` y `Obligatorio`, no tienen ninguna restricción y no depende de otro campo. Otro ejemplo es el campo `Valor Predeterminado`, este campo tiene algunas opciones por ingresar como posibles valores

(Texto, Fecha o `True`/`False`) sin embargo eso dependerá del Tipo de Dato que se seleccione, si se selecciona el tipo de dato `d:int` entonces el posible valor será Texto con la restricción de ingreso de sólo números enteros.

Tabla 3.1 Posibles valores por ingresar en una Propiedad (Parte 1 de 2)

Campo	Posibles valores	Restricción	Depende de
Nombre	Texto	Sólo números, letras, guiones (-) o guiones bajos (_) solamente	-
Título	Texto	Sin restricción	-
Descripción	Texto	Sin restricción	-
Tipo de Dato	Opción: `d:text`	Sin restricción	-
	Opción: `d:mltext`	Sin restricción	-
	Opción: `d:int`	Sin restricción	-
	Opción: `d:float`	Sin restricción	-
	Opción: `d:double`	Sin restricción	-
	Opción: `d:date`	Sin restricción	-
	Opción: `d:boolean`	Sin restricción	-
	Opción: `d:long`	Sin restricción	-
Requerido	Opción: `Opcional`	Sin restricción	-
	Opción: `Obligatorio`	Sin restricción	-
Valor predeterminado	Texto	Sin restricción	`Tipo de Dato` (`d:text`), `Requerido` y `Restricción`
	Texto	Sin restricción	`Tipo de Dato` (`d:mltext`), `Requerido` y `Restricción`
	Texto	Sólo números enteros	`Tipo de Dato` (`d:int`), `Requerido` y `Restricción`
	Texto	Sólo números, incluye punto decimal	`Tipo de Dato` (`d:float`), `Requerido` y `Restricción`
	Texto	Sólo números, incluye punto decimal	`Tipo de Dato` (`d:double`), `Requerido` y `Restricción`
	Fecha	Sólo fechas	`Tipo de Dato` (`d:date`), `Requerido` y `Restricción`

Tabla 3.1 Posibles valores por ingresar en una Propiedad (Parte 2 de 2)

Valor Predeterminado	`True` / `false`	Sólo true o false	`Tipo de Dato` (`d:boolean`), `Requerido` y `Restricción`
	Texto	Sólo números enteros	`Tipo de Dato` (`d:long`), `Requerido` y `Restricción`
Restricción	Opción: `Expresión Regular`	Sólo expresiones regulares	`Tipo de Dato` (`d:text` y `d:mltext`)
	Opción: `Longitud mínima/máxima`	Sin restricción	`Tipo de Dato` (`d:text` y `ml:text`)
	Opción: `Valor mínimo/máximo`	Sólo números	`Tipo de Dato` (`d:int`, `d:long`, `d:float` y `d:double`)
Indexación	Opción: `Ninguna`	Sin restricción	-
	Opción: `Texto Libre`	Sin restricción	-

A continuación, se muestran algunas de las validaciones realizadas.

La primera validación se realiza cuando en el campo `Nombre` se intenta ingresar caracteres no permitidos. El mensaje de error se muestra en la Figura 3.66.

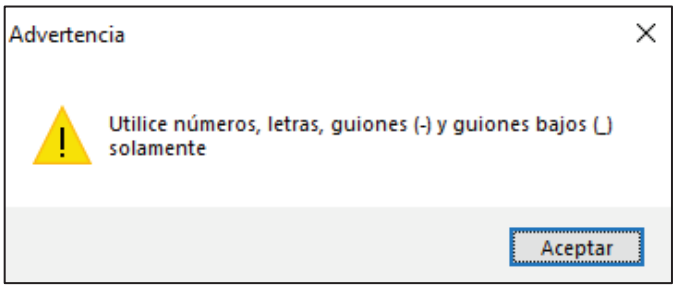


Figura 3.66 Mensaje de error al intentar ingresar caracteres no permitidos

Si el valor del campo `Requerido` es `Obligatorio` entonces no será posible ingresar un valor predeterminado, debido a que el valor `Obligatorio` obliga al usuario a ingresar un valor de metadato. El valor predeterminado solo se aplica cuando el campo `Requerido` es `Opcional` En la Figura 3.67 se muestra como el campo Valor Predeterminado se bloquea cuando la opción es Obligatorio.

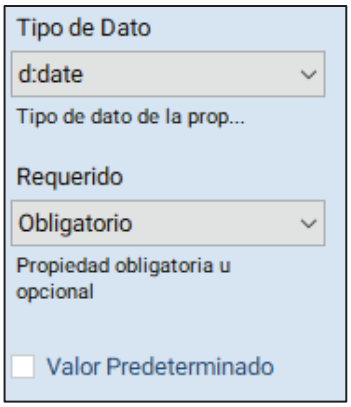


Figura 3.67 `Valor Predeterminado` bloqueado por `Requerido Obligatorio`

Si se quiere ingresar una restricción Expresión Regular entonces el tipo de dato debe estar seleccionado en `d:text` o `d:mltext`. Si se ha especificado un tipo de dato diferente, entonces al momento de intentar seleccionar la restricción Expresión Regular se mostrará el mensaje de error de la Figura 3.68

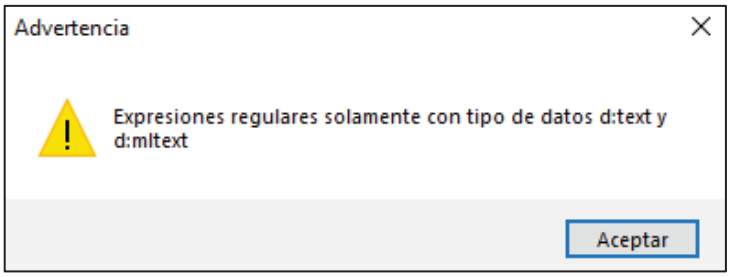


Figura 3.68 Mensaje de error al intentar ingresar la restricción Expresión regular

Si en el campo Expresión Regular se ingresa una expresión regular no válida, luego de accionar el botón `Crear/Editar` se abortará la creación o edición y se mostrará el mensaje de la Figura 3.69.

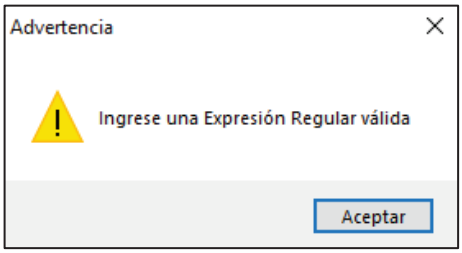


Figura 3.69 Mensaje de error tras intentar ingresar una Expresión Regular no válida

La restricción Longitud mínima/máxima se aplica solamente a los tipos `d:text` y `d:mltext`, si esta restricción se intenta aplicar a otro tipo diferente entonces al accionar el botón Crear/Editar se abortará la transacción y se mostrará el mensaje de la . Esta

restricción está enlazada al valor del campo `Valor Predeterminado`, en caso de este no sea de la longitud especificada en la restricción (Figura 3.71) entonces se mostrará el mensaje de error de la Figura 3.72.

Figura 3.70 Error. La restricción de longitud solo se aplica a los tipos `string`

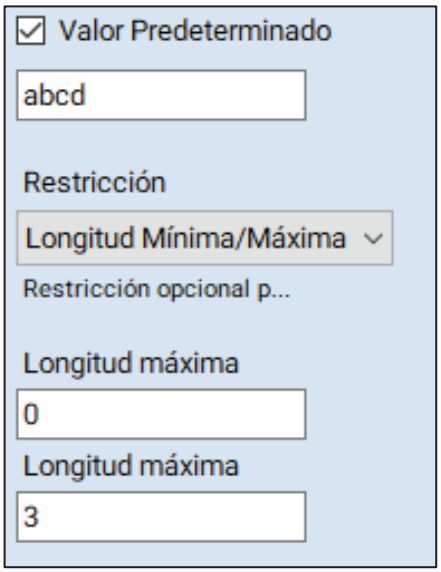


Figura 3.71 Restricción Longitud Mínima/Máxima

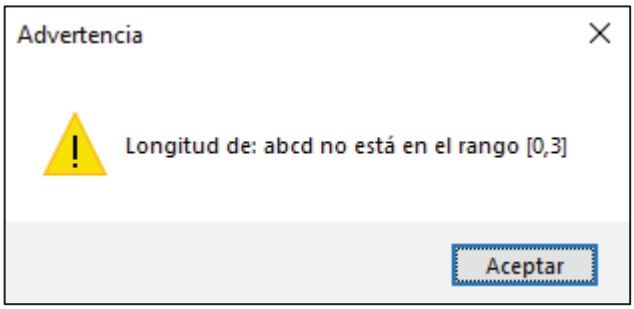


Figura 3.72 Error. La restricción de longitud no coincide con el valor predeterminado

La restricción para valores mínimos y máximos solo se aplica para tipos numéricos, en caso de que el tipo de dato no sea numérico entonces se mostrará el mensaje de la Figura 3.73. En el caso de la Figura 3.74 muestra la restricción de valor mínimo y máximo con

valor mínimo es 0 y el valor máximo es 10. Si el Valor Predeterminado está fuera de rango de valores mínimo y máximo entonces se muestra el mensaje de error de la Figura 3.75.

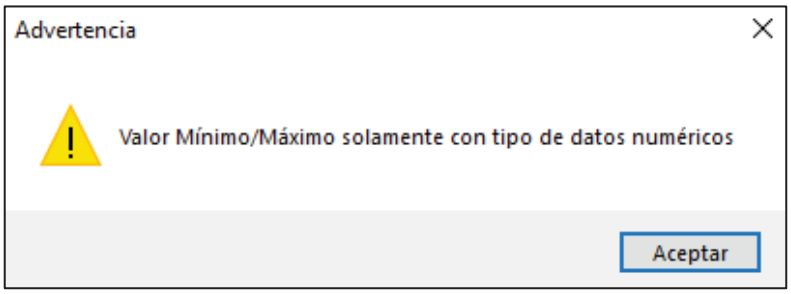


Figura 3.73 Mensaje de error cuando el tipo de dato no es numérico

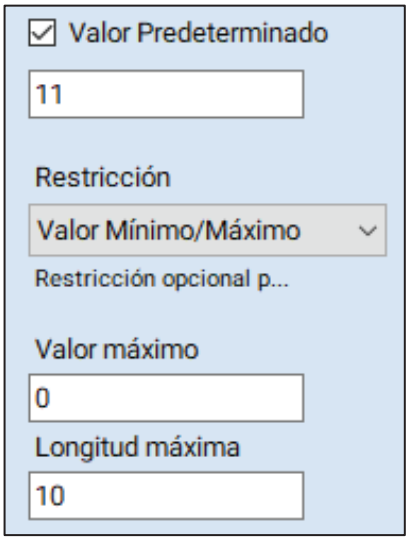


Figura 3.74 Restricción Valor Mínimo/Máximo

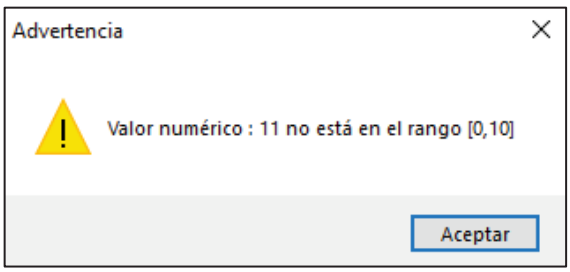


Figura 3.75 Mensaje de error cuando el valor predeterminado está fuera de rango

3.1.12 Agregar y almacenar sub-repositorios de Alfresco en la PC

Para validar este requerimiento se selecciona un sub-repositorios de Alfresco para que este pueda ser sincronizado con la computadora local. Este requerimiento se lo validó desde el formulario de sincronización mostrado en la Figura 3.76.

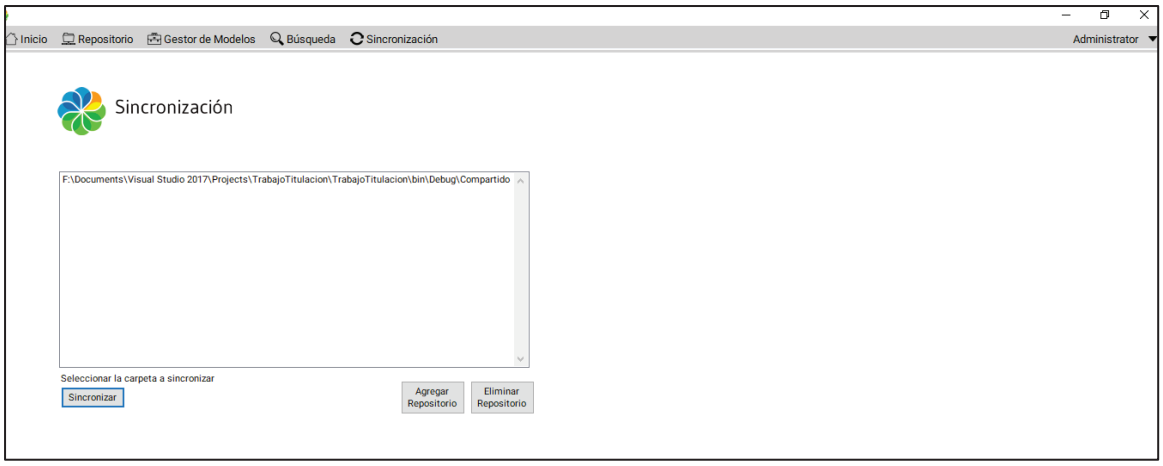


Figura 3.76 Formulario de sincronización

Para sincronizar un sub-repositorio es necesario hacer clic en el botón `Agregar Repositorio`. Esta acción permitirá seleccionar un sub-repositorio de la lista mostrada en la Figura 3.77.

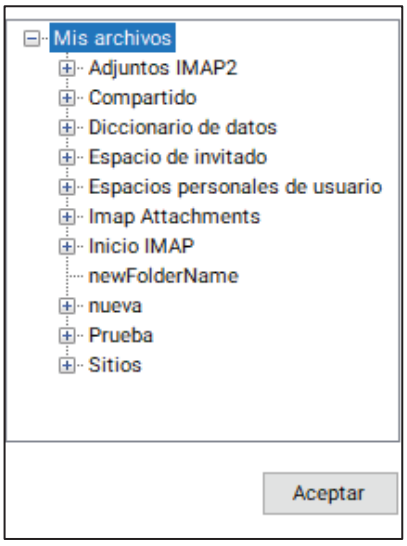


Figura 3.77 Control `TreeView` para seleccionar un sub-repositorio a sincronizarse

Una vez que se seleccione el sub-repositorio entonces automáticamente este se replicará sobre la carpeta donde se está ejecutando el prototipo, en este caso la carpeta `bin/DEBUG`. También la ubicación del sub-repositorio replicado se guardará en el archivo de configuración del prototipo. Se ha seleccionado el sub-repositorio `Compartido` de la Figura 3.77 y se ha hecho clic sobre el botón `Aceptar`. Se ha creado un repositorio local con el contenido de la carpeta compartido, este repositorio se muestra en la Figura 3.78.

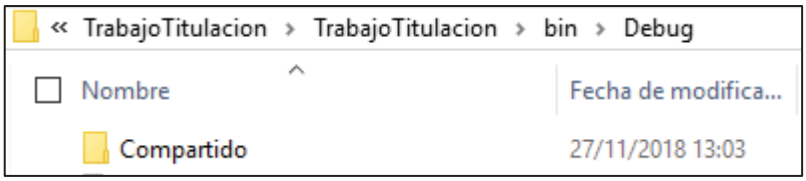


Figura 3.78 Carpeta local con el sub-repositorio replicado

3.1.13 Sincronización de los contenidos de Alfresco con la PC

Este requerimiento se lo validó desde el formulario de la Figura 3.76. Se seleccionó un sub-repositorio agregado anteriormente y se hizo clic sobre el botón `Sincronizar`. Esta acción desencadenó que los cambios realizados sobre los archivos o carpetas locales se repliquen en el servidor de Alfresco y viceversa.

En la Tabla 3.2 se registra las validaciones realizadas. A parte de ello se muestra una constancia de un archivo modificado en la PC y tal modificación se muestra en Alfresco Share.

Tabla 3.2 Validación de cada acción que requiere la sincronización

Acción	¿Se sincroniza?				
	Carpeta Nivel 1	Carpeta Nivel 2	Carpeta Nivel 5	Archivos Office	Archivos comunes
El usuario crea un archivo/carpeta en la computadora	Sí	Sí	Sí	Sí	Sí
El usuario modifica un archivo/carpeta en la computadora	Sí	Sí	Sí	Sí	Sí
El usuario elimina un archivo/carpeta en la computadora	Sí	Sí	Sí	Sí	Sí
El usuario crea un nodo en Alfresco	Sí	Sí	Sí	Sí	Sí
El usuario modifica un nodo en Alfresco	Sí	Sí	Sí	Sí	Sí
El usuario elimina un nodo en Alfresco	Sí	Sí	Sí	Sí	Sí

Cuando se modifica el archivo `ArchivoCreadoDesdePC.txt` desde la PC, como se muestra en la Figura 3.79 (se le agregó el texto mostrado), entonces los cambios se ven reflejados en el archivo símil de Alfresco Share. En la Figura 3.80, se muestra el archivo `ArchivoCreadoDesdePC.txt` sincronizado, la captura es desde Alfresco Share.

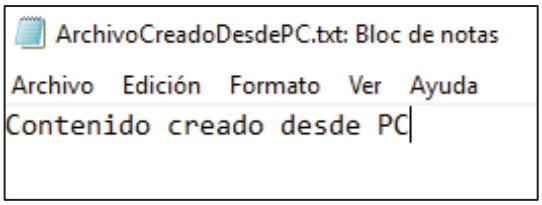


Figura 3.79 Modificación de archivo de la PC sincronizado

Figura 3.80 Archivo sincronizado, visto desde Alfresco Share

3.1.14 Búsqueda de contenidos basada en metadatos

Este requerimiento se lo validó desde el formulario de la Figura 3.81. Se realizó varias pruebas y modificando los parámetros de este gestor. Se buscó por Tipo, por Aspecto, por Propiedad y por el valor de una Propiedad, cuando se busca por el valor de una Propiedad se modificó los diferentes controles que permiten realizar comparaciones.

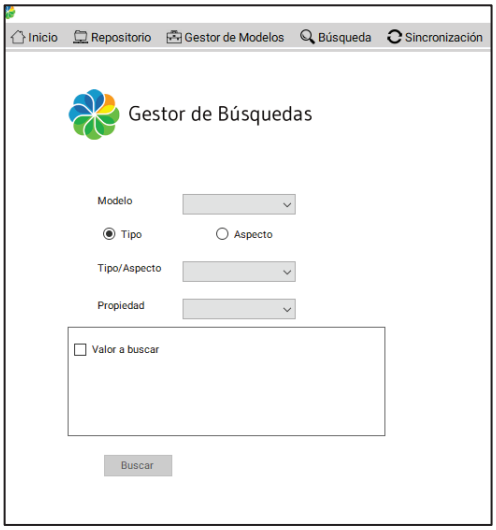


Figura 3.81 Gestor de búsquedas

Primero, se realiza una búsqueda por Aspecto. El Aspecto buscado es `Sincronizable`. La Figura 3.82, muestra que se ha seleccionado el Aspecto `Sincronizable`, a

continuación, se hizo clic en el botón `Buscar` y los resultados se muestran en la parte derecha del formulario.

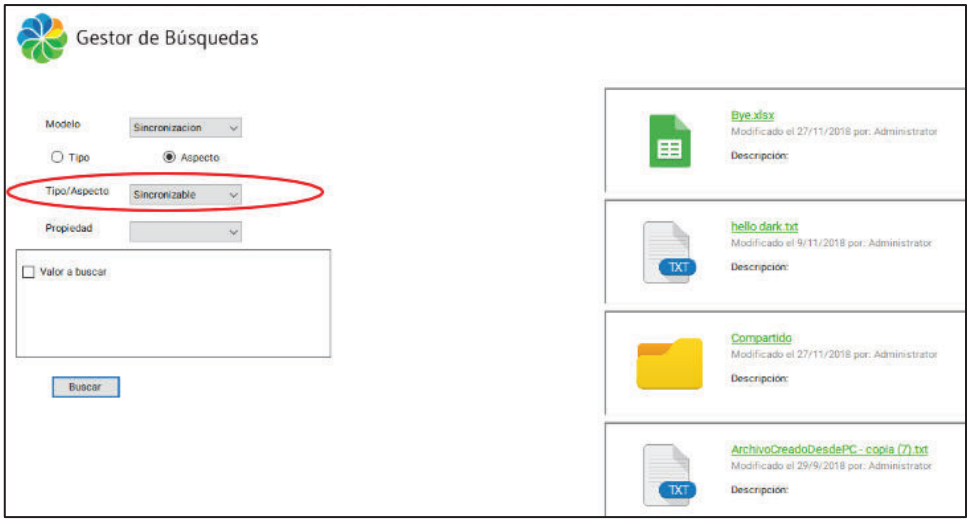


Figura 3.82 Resultados de la búsqueda por Aspecto `Sincronizable`

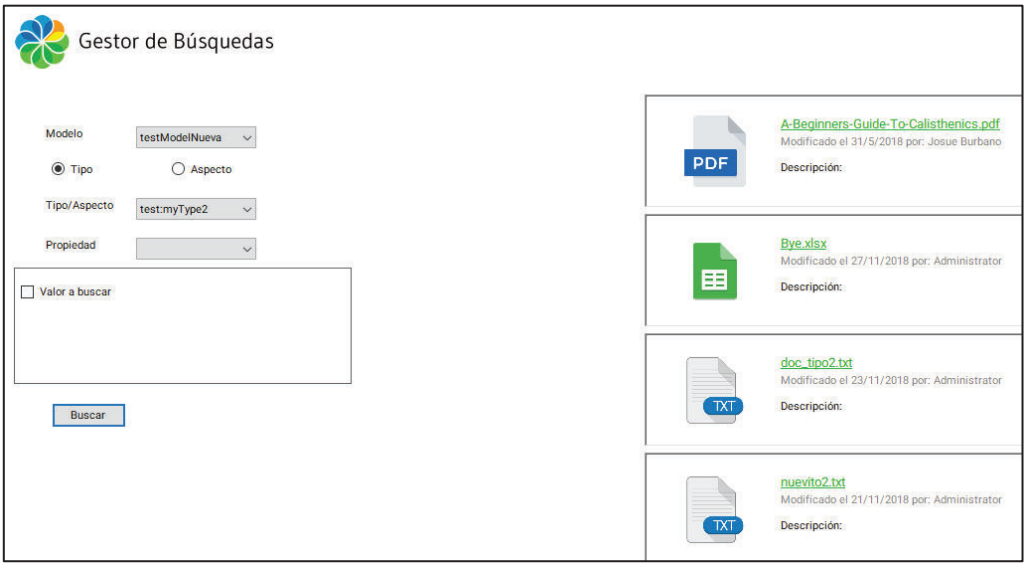


Figura 3.83 Resultados de la búsqueda por Aspecto `Sincronizable`

Se realiza una búsqueda por Tipo. El Tipo buscado es `test:myType2`. La Figura 3.84, muestra que se ha seleccionado el Tipo `test:myType2`. Los resultados se observan en la parte derecha del formulario.

Se realiza una búsqueda por Propiedad. La Propiedad buscada es `Sincronizable`. La Figura 3.84, muestra que se ha seleccionado la Propiedad `Sincronizable` y después de hacer clic en el botón `Buscar` se accionó la búsqueda. Los resultados se muestran en la parte derecha del formulario.

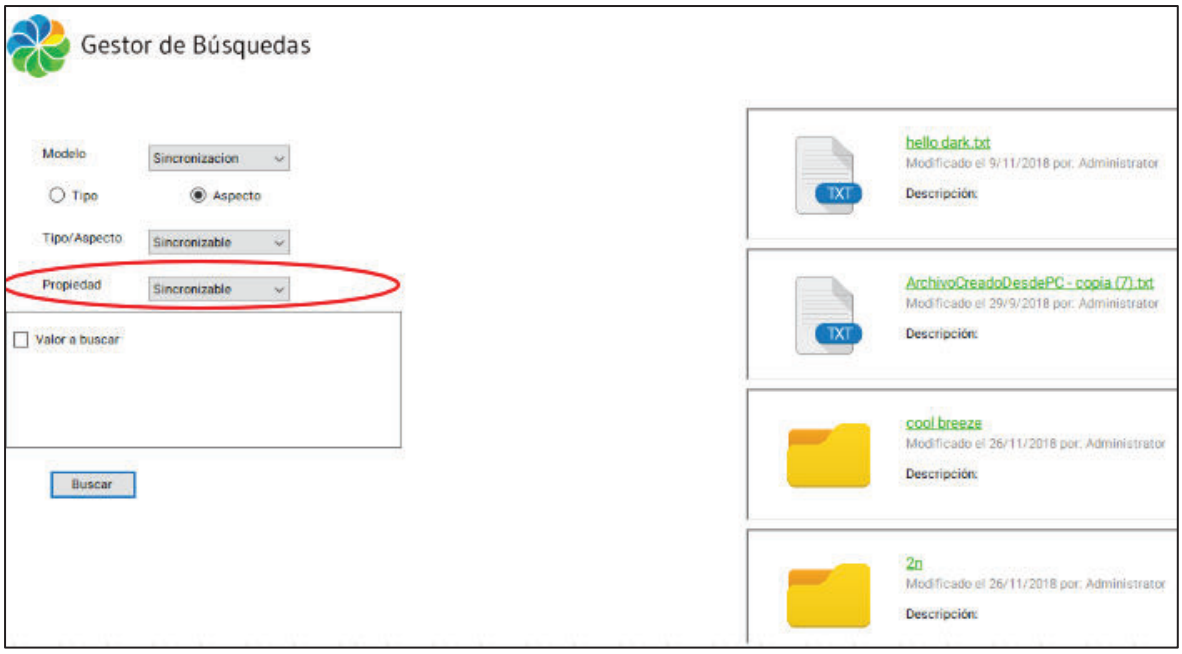


Figura 3.84 Resultados de la búsqueda por Propiedad `Sincronizable`

Si se busca una Aspecto, un Tipo, una Propiedad o un valor que no coincida entonces se mostrará un mensaje en letras rojas indicando que los parámetros indicados no coinciden con la búsqueda. El mensaje se presenta en la parte inferior del botón `Buscar` como se muestra en la Figura 3.85.

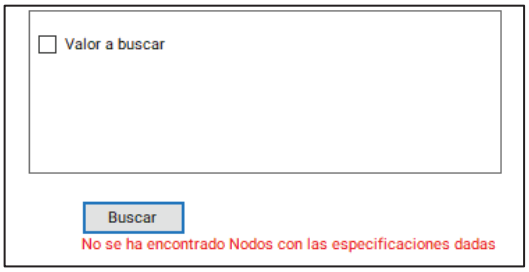


Figura 3.85 Parte del formulario cuando no se ha encontrado ningún Nodo

3.2 Pruebas de aceptación

Para las pruebas de aceptación realizó una encuesta que permite validar la funcionalidad de la aplicación. La encuesta cuenta con diez preguntas y fue realizada a veinte usuarios de Alfresco. El modelo de encuesta se adjunta en el ANEXO IV.

La primera pregunta fue ¿es posible iniciar sesión en la aplicación con las credenciales de Alfresco?' y se la realizó con el fin de validar el acceso a la aplicación con las credenciales de Alfresco. Los resultados se muestran en la Figura 3.86 Los resultados fueron del 100% de aceptación.

Figura 3.86 Resultados de la pregunta 1

La segunda pregunta fue ¿es posible navegar entre el Navegador de Repositorio y la búsqueda de Contenidos en la aplicación? Y se la realizó el fin de comprobar la navegación entre los módulos de Navegación de Repositorio y búsqueda de Contenidos en la Aplicación. Los resultados se muestran en la Figura 3.87. De igual manera, los resultados fueron 100% por la respuesta sí.

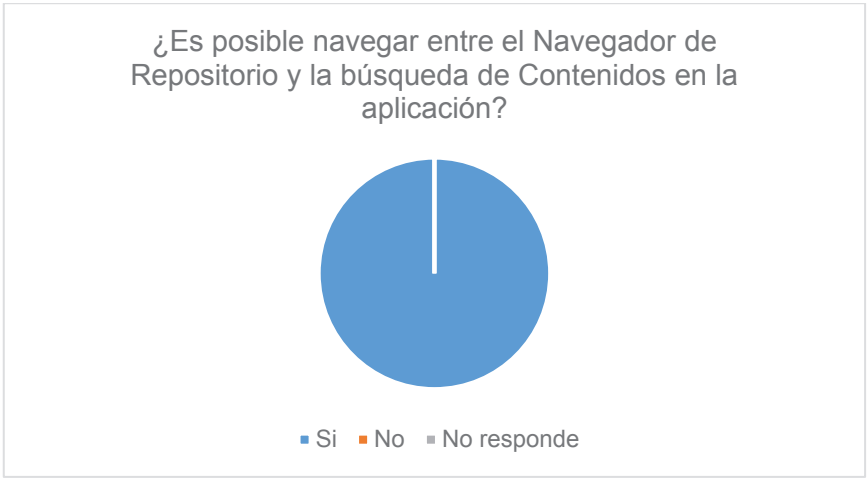


Figura 3.87 Resultados de la pregunta 2

La tercera pregunta fue ¿si se modifica un archivo en el Navegador de Repositorio de la aplicación, el cambio se ve reflejado en Alfresco Share? se la realizó con el fin de comprobar persistencia en el caso de modificar un archivo en el Navegador de Repositorio de la aplicación con Alfresco Share. Los resultados se muestran en la Figura 3.88. Un 10% respondió no a esta pregunta, ellos fueron los dos primeros encuestados, luego de ello se corrigió el error.

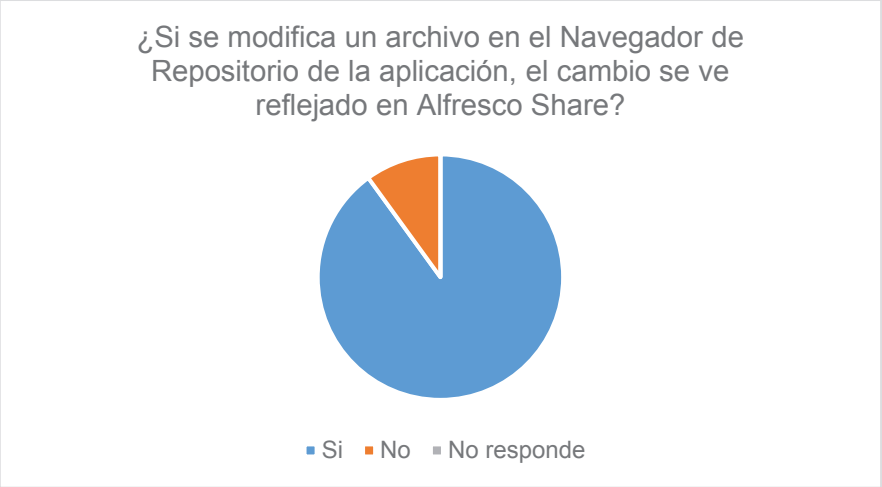


Figura 3.88 Resultados de la pregunta 3

La cuarta pregunta fue ¿es posible cargar archivos y carpetas masivamente a través de arrastrar y soltar? Se la realizó con el fin de comprobar si se puede cargar masivamente archivos. Los resultados se muestran en la Figura 3.89 y muestran que un 10% de los encuestados no pudieron arrastrar y soltar, mientras que el 90% restante sí.

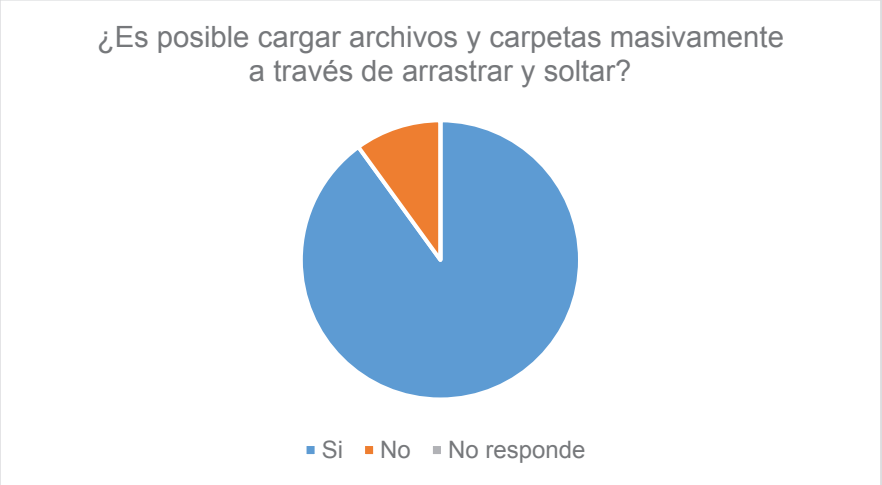


Figura 3.89 Resultados de la pregunta 4

La quinta pregunta fue se ha definido el Tipo `alfepn:Prueba` con los metadatos `alfepn:Metadato_prueba` y `alfepn:Metadato2_prueba` ¿Es posible aplicar esos metadatos a un a un archivo? se la realizó con el fin de verificar si es posible cambiar de Tipo a un contenido. Los resultados se muestran en la Figura 3.90. El 100% de los encuestados está de acuerdo con esta funcionalidad.

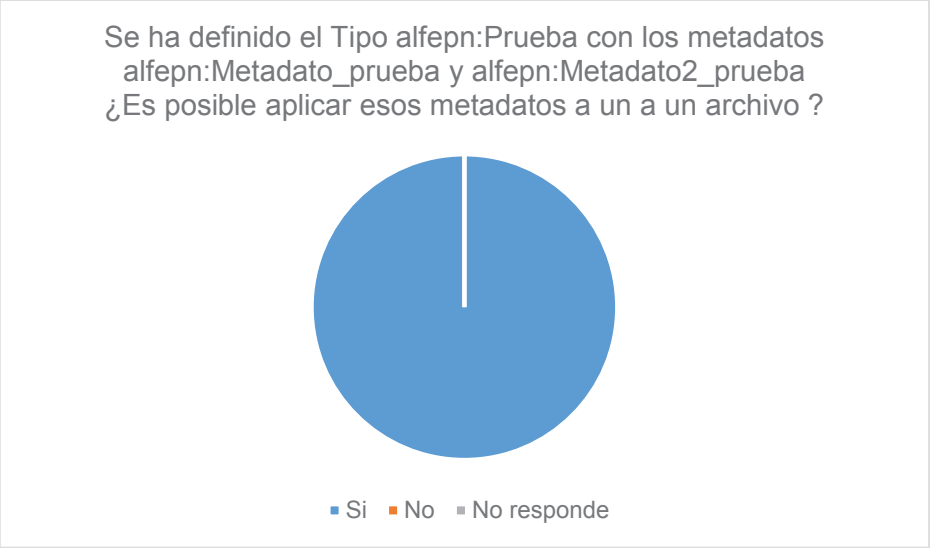


Figura 3.90 Resultados de la pregunta 5

La sexta pregunta fue ¿es posible añadir un valor al metadato `alfepn:Metadato2_prueba`? Se la realizó con el fin de validar si es posible editar y ver metadatos sobre un contenido Los resultados se muestran en la Figura 3.91 y muestran que el 100% de los encuestas está de acuerdo con esta pregunta.

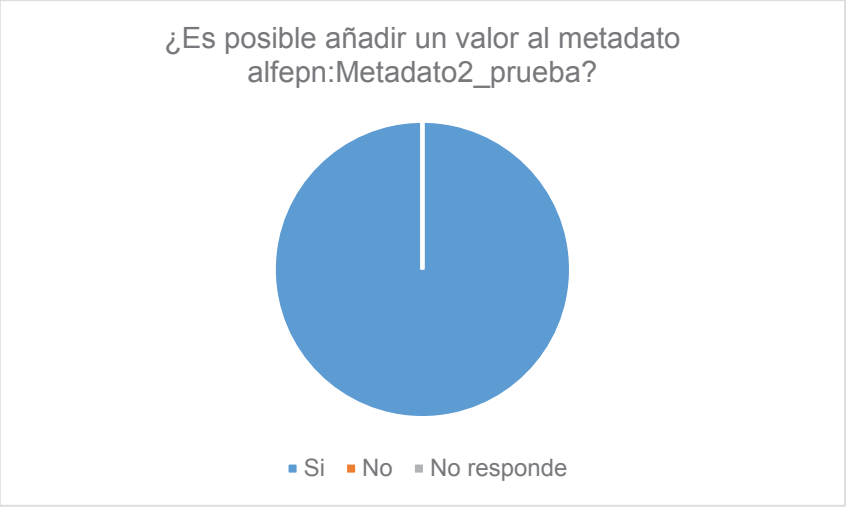


Figura 3.91 Resultados de la pregunta 6

La séptima pregunta fue ¿cuándo se modifica un archivo en el repositorio local de la PC y luego de sincronizar, el cambio se ve reflejado en Alfresco Share? Esta pregunta se la realizó con el fin de verificar sincronización. Los resultados están en la Figura 3.92 y muestran que el total de los encuestados está de acuerdo.

La octava pregunta ¿Cuándo se modifica un archivo en Alfresco Share y luego de sincronizar, el cambio se ve reflejado en el repositorio local de la PC? Se la realizó con el

objetivo de validar la sincronización. Los resultados se muestran en la Figura 3.93. El 100% de los encuestados marcó sí.

Figura 3.92 Resultados de la pregunta 7

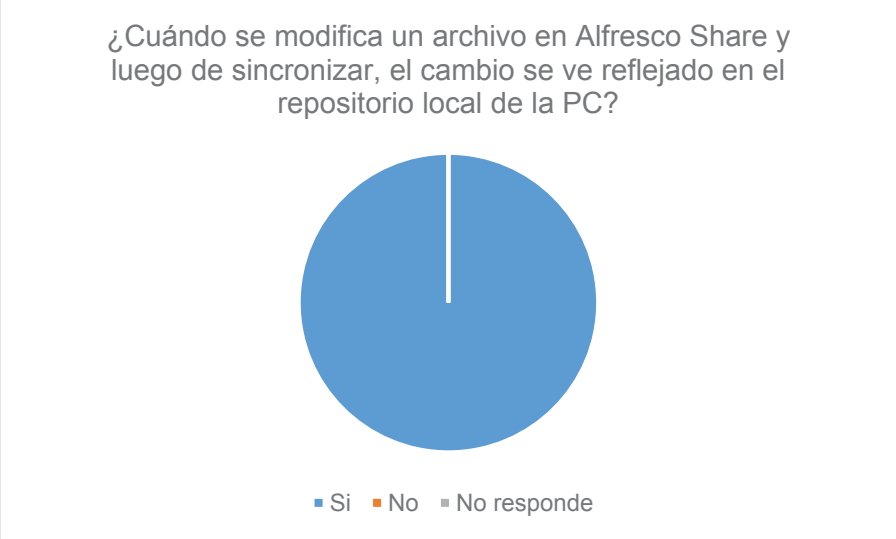


Figura 3.93 Resultados de la pregunta 8

La novena pregunta ¿Es posible buscar el metadato `alfepn:Metadato2_prueba` según el título del metadato? Se ha realizado con el fin de validar la búsqueda basada en metadatos Los resultados se muestran en la Figura 3.94. El 100% de los encuestados logró validar esta pregunta.

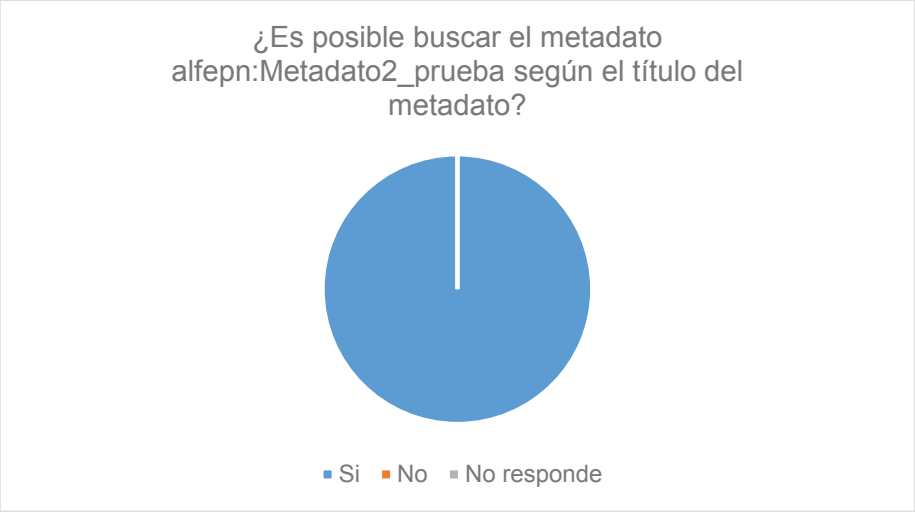


Figura 3.94 Resultados de la pregunta 9

La décima pregunta ¿Es posible buscar el metadato `alfepn:Metadato2_prueba` según el valor del metadato? Se ha realizado con el fin de validar la búsqueda basada en metadatos. Los resultados se muestran en la Figura 3.95. El 10% no estuvo en acuerdo con esta pregunta. Debido a que la coincidencia por corrección exacta y la coincidencia por error ortográfico no estaban habilitadas. Luego de ello, se habilitó tales características para obtener aceptación.

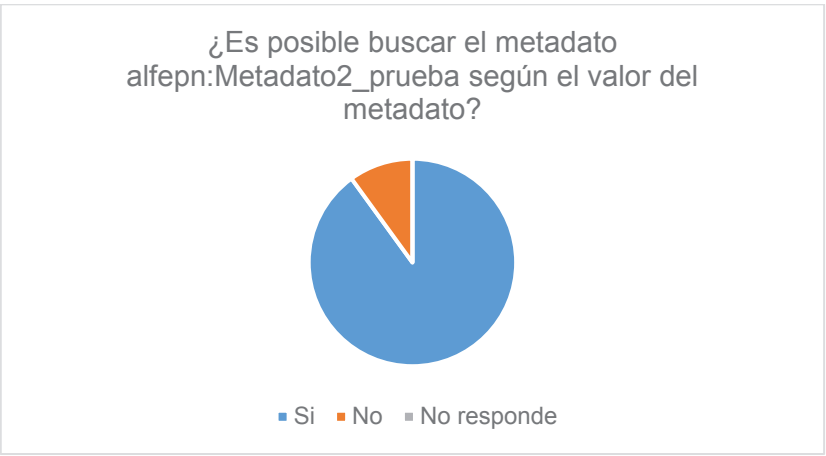


Figura 3.95 Resultados de la pregunta 10

3.3 Corrección de errores

El primer error que se detectó fue la demora en la carga del repositorio. A pesar de contar con un repositorio ligero la carga fue demasiada demorada. Lo que se decidió realizar es evitar cargar todo el repositorio directamente sobre la aplicación e irlo cargando por niveles, según la selección del usuario. En un inicio se cargaba todo el repositorio y se lo

almacenaba en RAM, luego solo se cargó inicialmente el primer nivel, luego si otro nivel fuera seleccionado entonces se cargó tal nivel y así sucesivamente.

Otro error que se detectó y corrigió fue en el formulario donde se muestran y editan los metadatos de archivos y carpetas. En un inicio todos los valores de las Propiedades, sin importar su tipo se mostraban y modificaban sobre el control `TextBox`. Sin embargo, un tipo de datos `Datetime` tiene su propio control para mostrar y editar valores `Datetime`. Es por ello que se filtró las Propiedades del tipo `d:datetime` o `d:date` para que sus valores sean presentados y puedan ser modificados en un `DateTimePicker`.

Otro error que se detectó fue la separación entre el módulo Sincronización y el módulo Navegador de Repositorio. En un inicio los cambios que se realizaban sobre archivos y carpetas de usuario utilizando el Navegador de Repositorio, no se veían reflejados directamente en el repositorio local solo en el servidor de Alfresco. Esto implicó que los cambios que se realicen sobre el módulo de Navegador de Repositorio automáticamente se vean reflejados tanto en el servidor como en el repositorio local.

Se detectaron errores relacionados con los permisos de usuario, debido a que no todos los usuarios pueden crear Modelos de Contenido o no todos los usuarios pueden editar todos los contenidos (debido a que no son los dueños o no cuentan con los permisos). Esto se corrigió usando excepciones que dependen del código de estado y solicitando la lista de los usuarios con permisos.

Además, fue posible identificar errores al momento de la ejecución en plataformas x64, esto fue debido a que la librería que edita los metadatos de los archivos locales no estaba registrada como dependencia para la plataforma x64, solo para la plataforma x86.

En esta sección también se decidió añadir cierta funcionalidad que hace más manejable al prototipo. Primero, se decidió colocar iconos a cada contenido dependiendo de su tipo de datos tanto para que se muestren en el repositorio como para cuando se editen las propiedades de los mismos. Luego, se decidió activar los botones de descarga de cada contenido para que estos puedan ser descargados en alguna carpeta local, las descargas funcionan tanto para archivos como para carpetas.

Errores derivados de las pruebas de usuario como falta de controles más descriptivos o errores generados en el accionar del usuario, por ejemplo, errores al accionar botones debido a que la selección era nula o errores debido a que luego de realizar una acción no se limpiaban los controles, fueron corregidos en este apartado.

Se corrigieron errores en la interfaz gráfica de usuario. Se optimizó espacios, se reubicó controles para obtener una mejor presentación de los resultados.

Cuando se arrastra y suelta archivos, estos no estaban siendo sincronizados automáticamente. Se programó tal automatización.

Al momento de cerrar la aplicación se programó para que se detecten si se han realizado cambios y no se han sincronizado. Se programó un mensaje de advertencia que permite guardar los cambios no sincronizados.

4 CONCLUSIONES Y RECOMENDACIONES

En esta sección se incluye el análisis de los resultados obtenidos en forma de conclusiones y recomendaciones.

4.1 Conclusiones

- Se dispone de un prototipo de aplicación distribuida que permite gestionar contenidos, basado en Alfresco para la Escuela Politécnica Nacional.
- Se cuenta con un prototipo que ha integrado los servicios de gestión y aplicación de metadatos, visualización de contenidos, búsqueda basada en metadatos y sincronización de contenido.
- El módulo gestor de metadatos permite leer, crear, modificar y eliminar modelos de metadatos.
- A través del módulo de búsqueda es posible buscar contenidos por Aspectos, Tipos, Propiedades y valores de Propiedades.
- A través del módulo visualizador de archivos es posible visualizar archivos y carpetas almacenados en Alfresco, así como visualizar, agregar, eliminar o editar valores de metadatos sobre tales archivos.
- A través del módulo para sincronización es posible verificar cambios sobre los contenidos de un repositorio local y actualizar tales cambios sobre su repositorio símil en Alfresco. De igual manera es posible verificar cambios sobre los contenidos de un repositorio remoto de Alfresco y actualizar tales cambios sobre su repositorio local símil en la PC de usuario.
- Para la consecución de este prototipo fue necesario aprender sobre el funcionamiento de Alfresco, su sistema de meta-gestión y sus interfaces de comunicación, en particular sobre sus API RESTful.
- Se encontró que el esquema de comunicación más utilizado para clientes Alfresco es el esquema basado en el protocolo CMIS. Sin embargo, este prototipo se desarrolló sobre las API RESTful de Alfresco debido a que el soporte para gestión de metadatos está mucho más desarrollado para las API.
- La estructura de los archivos Swagger de cada API permitió contar con un modelo de dominios apegado a las acciones de cada API y a partir del mismo fue posible

construir otros modelos de dominio que sigan la estructura de las API oficiales de Alfresco.

- La manipulación de metadatos sobre archivos locales permitió emular una base de datos, la cual estaría diseñada para almacenar datos que permitan sincronización. Los metadatos sobre los archivos locales fueron suficientes para completar la sincronización y así se evitó tener procesos de bases de datos corriendo en segundo plano.

- El uso de una herramienta que permita versionar y realizar respaldos del código, permitió tener un desarrollo fluido, en algún punto los archivos de configuración no funcionaron o las clases contenían errores luego de una refactorización y fue el uso de Git el que permitió salir de ello rápidamente.

- Aunque algunos de los módulos de esta aplicación ya se encuentran implementados en otros clientes, la idea de este prototipo es tener una integración de servicios de manera que el control de los contenidos, así como la gestión de metadatos estén en un mismo sitio y de esta manera facilitar el trabajo al usuario.

- Aunque Alfresco Share posee la capacidad de gestionar modelos de contenido, este prototipo aventaja a Alfresco Share porque permite que los usuarios no administradores puedan explorar la estructura de los metadatos y así tener una estructura menos descentralizada que permita sugerir cambios sobre los modelos.

- El trabajar con un esquema asíncrono permitió manejar un esquema de recepción adecuado de la información remota. Esto permitió crear formulario de carga y evitó que la aplicación se bloquee cuando una solicitud es demasiado demorada.

- El trabajar con un esquema asíncrono tuvo desventajas a la hora de depuración. Cuando una excepción se produce, el origen de la excepción se ubicará en el hilo principal y no en el que se está ejecutando en segundo plano. Por lo que identificar el error se volvió demoroso.

- Utilizar herramientas como Wireshark o Postman al momento de trabajar con API se volvió indispensable. Estas herramientas se utilizaron para aprender de las solicitudes generadas por el cliente Alfresco Share y replicar el comportamiento en la aplicación. Al momento de verificar errores y realizar pruebas de integración estas herramientas fueron imprescindibles.

- La estructura de archivos y carpetas del código del prototipo permitió manipular y verificar fácilmente el código, además el esquema es tal que perfectamente está estructurado para posible escalamiento del prototipo.

4.2 Recomendaciones

- Se recomienda seguir trabajando sobre el prototipo de tal manera que al final se pueda conseguir una total integración de servicios de Alfresco y servicios de otros gestores de contenidos como Share Point. Además, este nuevo prototipo podría estar conectado al sistema de gestión de autenticación de la EPN para que con el mismo usuario y contraseña de las credenciales institucionales se acceda a servicios de gestión de contenido.

- El proceso de sincronización podría ser trabajado para que se ejecute en segundo plano (a través de un demonio que este escuchando constantemente) y automáticamente permita verificar cambios locales y remotos, y actualizarlos correspondientemente.

- El proceso de sincronización podría ser programado por el usuario para determinado tiempo, es decir, que la aplicación se sincronice automáticamente cada 1, 2, 3 o 30 minutos.

- Se recomienda crear un archivo Swagger con la información de la API ContentCMN, para que la API se vuelva oficial y la implementación de cualquier cliente pueda ser replicada fácilmente en los diferentes lenguajes de programación.

- Para posteriores implementaciones se recomienda diseñar, previo a la implementación, un esquema para validación y errores el cual permita encajar en el código sin alterar su estructura normal. Además, las pruebas podrían ser automatizadas a través de *scripts* para evitar verificar a través de solicitudes `GET`.

- Se recomienda utilizar un esquema o planificación que permitan documentar mientras se va implementando de tal manera que cuando cada *sprint* haya terminado poder contar con diseño, implementación, retrospectiva y pruebas documentadas.

- Se recomienda verificar el uso de API mediante llamadas HTTPS, debido que todo este Trabajo se lo ha realizado mediante HTTP sin cifrado.

- Se recomienda unir los módulos Sincronización y Navegador de Repositorio, debido a que estos dos módulos pueden trabajar mucho mejor juntos.

5 REFERENCIAS BIBLIOGRÁFICAS

[1] I. Alfresco, “Content Model | Alfresco Documentation.” [Online]. Available: https://docs.alfresco.com/community/references/dev-extension-points-content-model.html. [Accessed: 01-Jun-2018].

[2] M. van Steen and A. S. Tanenbaum, “A brief introduction to distributed systems,” *Computing*, vol. 98, no. 10, pp. 967–1009, 2016.

[3] A. S. Tanenbaum and D. J. Wetherall, *Redes De Computadoras*. 2012.

[4] I. Sommerville, *Software Engineering*, 9th ed. Pearson, 2010.

[5] R. Fielding *et al.*, “RFC2616 - Hypertext transfer protocol–HTTP/1.1,” *Internet Eng. Task Force*, pp. 1–114, 1999.

[6] M. Anicas, “How To Troubleshoot Common HTTP Error Codes,” *How To Troubleshoot Common HTTP Error Codes*, 2014. [Online]. Available: https://www.digitalocean.com/community/tutorials/how-to-troubleshoot-common-http-error-codes. [Accessed: 21-May-2018].

[7] B. Guze, “HTTP Headers for Dummies,” *HTTP Headers for Dummies*, 2009. [Online]. Available: https://code.tutsplus.com/tutorials/http-headers-for-dummies--net-8039. [Accessed: 23-May-2018].

[8] D. Jacobson, D. Woods, and G. Brail, *APIs: A Strategy Guide*, 1st ed. O’Really Media, 2012.

[9] Oracle, “Java JDBC API.” [Online]. Available: https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/. [Accessed: 28-May-2018].

[10] Twitter, “Tapinto what’s happening.” [Online]. Available: https://developer.twitter.com/content/developer-twitter/en.html. [Accessed: 28-May-2018].

[11] L. Richardson and S. Ruby, *RESTful Web Services*. 2007.

[12] L. Richardson, M. Amundsen, and S. Ruby, *RESTful Web APIs*, 1st ed. O’Really Media, 2013.

[13] “Introducing JSON.” [Online]. Available: https://www.json.org/. [Accessed: 29-May-2018].

[14] “Alfresco Community Edition 201804 EA,” *Alfresco Documentation*. [Online]. Available: https://docs.alfresco.com/community/concepts/welcome-infocenter_community.html.

[15] T. Allen, “The Importance of Metadata in Content Management,” 2011. [Online]. Available: https://www.cmswire.com/cms/enterprise-cms/the-importance-of-metadata-in-content-management-009746.php. [Accessed: 01-Jun-2018].

[16] C. Riley, “How important is document metadata in ECM systems?,” 2010. [Online]. Available: https://searchcontentmanagement.techtarget.com/answer/How-important-is-document-metadata-in-ECM-systems. [Accessed: 01-Jun-2018].

[17] J. Potts, “Working With Custom Content Types in Alfresco | ECM Architect | Alfresco Developer Tutorials,” 2018. [Online]. Available: http://ecmarchitect.com/alfresco-developer-series-tutorials/content/tutorial/tutorial.html. [Accessed: 01-Jun-2018].

[18] I. Alfresco, “Content modeling with Model Manager | Alfresco Documentation.” [Online]. Available: https://docs.alfresco.com/5.2/concepts/admintools-cmm-intro.html. [Accessed: 06-Jun-2018].

[19] W3schools, “XML Namespaces.” [Online]. Available: https://www.w3schools.com/xml/xml_namespaces.asp. [Accessed: 05-Jun-2018].

[20] I. Alfresco, “Alfresco Content Services REST API Explorer.” [Online]. Available: https://api-explorer.alfresco.com/api-explorer/#/. [Accessed: 06-Jun-2018].

[21] E. Romano, “Alfresco JavaScript API Client.” [Online]. Available: https://github.com/Alfresco/alfresco-js-api#full-documentation-of-all-the-methods-of-each-api. [Accessed: 06-Jun-2018].

[22] SmartBear Software, “What is Swagger | Swagger,” 2018. [Online]. Available: https://swagger.io/docs/specification/2-0/what-is-swagger/. [Accessed: 16-Sep-2018].

[23] SCRUMstudy, “A Guide to the SCRUM BODY OF KNOWLEDGE (SBOK™ Guide) 2016 Edition A Comprehensive Guide to Deliver Projects using Scrum.”

[24] “Json.NET - Newtonsoft.” [Online]. Available: https://www.newtonsoft.com/json. [Accessed: 17-Sep-2018].

[25] RestSharp, “Getting Started with RestSharp,” 2017. [Online]. Available:

https://github.com/restsharp/RestSharp/wiki/Getting-Started.

[26] Microsoft, “The Dsofile.dll files lets you edit Office document properties when you do not have Office installed.” [Online]. Available: https://support.microsoft.com/en-us/help/224351/the-dsofile-dll-files-lets-you-edit-office-document-properties-when-yo. [Accessed: 10-Mar-2018].

[27] Microsoft, “Welcome to the Open XML SDK 2.5 for Office | Microsoft Docs.” [Online]. Available: https://docs.microsoft.com/en-us/office/open-xml/open-xml-sdk. [Accessed: 03-Oct-2018].

[28] E. Tomas, “C# 5: Async / Await.” [Online]. Available: https://geeks.ms/etomas/2011/09/17/c-5-async-await/. [Accessed: 04-Nov-2018].

[29] Microsoft, “Task-based Asynchronous Programming | Microsoft Docs.” [Online]. Available: https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming/task-based-asynchronous-programming. [Accessed: 05-Nov-2018].

[30] Microsoft, “await (C# Reference) | Microsoft Docs.” [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/await. [Accessed: 05-Nov-2018].

[31] Microsoft, “GitHub Extension for Visual Studio.” [Online]. Available: https://visualstudio.github.com/. [Accessed: 02-Nov-2018].

[32] F. V. Ochoa Carrión, “Desarrollo e implementación de un prototipo de seguridad web para la gestión y administración de documentos para el sistema Alfresco,” Apr. 2015.

[33] “Taiga.io.” [Online]. Available: https://taiga.io/. [Accessed: 09-Dec-2018].

[34] Microsoft, “Descargas | IDE, Code y Team Foundation Server | Visual Studio.” [Online]. Available: https://visualstudio.microsoft.com/es/downloads/. [Accessed: 02-Nov-2018].

[35] Alfresco Inc., “Installing Community Edition on Windows | Alfresco Documentation.” [Online]. Available: https://docs.alfresco.com/community5.1/tasks/simpleinstall-community-win.html. [Accessed: 02-Nov-2018].

[36] TechNet, “Visual Studio 2017: Install and Use GitHub Extension - TechNet Articles - United States (English) - TechNet Wiki.” [Online]. Available: https://social.technet.microsoft.com/wiki/contents/articles/38935.visual-studio-2017-

install-and-use-github-extension.aspx. [Accessed: 03-Nov-2018].

[37] Microsoft, “How to: Set a custom property in a word processing document (Open XML SDK) | Microsoft Docs.” [Online]. Available: https://docs.microsoft.com/en-us/office/open-xml/how-to-set-a-custom-property-in-a-word-processing-document. [Accessed: 15-Nov-2018].

[38] Alfresco Inc., “Alfresco Content Services REST API,” *Search API*. [Online]. Available: https://api-explorer.alfresco.com/api-explorer/#!/search/search. [Accessed: 14-Nov-2018].

[39] Alfresco Inc., “query | Alfresco Documentation.” [Online]. Available: https://docs.alfresco.com/6.0/concepts/search-api-query.html. [Accessed: 14-Nov-2018].

[40] “CmisSync: Dropbox-like sync for your enterprise file server.” [Online]. Available: https://cmissync.com/. [Accessed: 09-Dec-2018].

6 ANEXOS

En Anexos se muestran secciones que debido a su extensión no pueden ser incorporadas directamente en ninguna de las secciones anteriores.

ANEXO I. Modelo de encuesta para levantamiento de información

ANEXO II. Código de la aplicación

ANEXO III. Diagrama UML de clases e interfaces

ANEXO IV. Modelo de encuesta para pruebas de aceptación

ANEXO V. Manual de uso de la aplicación

ANEXO I

Modelo de encuesta para levantamiento de información

Instrucciones. Marque con X según su criterio

1. **¿De qué manera accede a Alfresco?**
   a. A través del navegador web ( )
   b. A través de un programa de escritorio ( ) Indique cual _____
   c. Otro ( ) Especifique_____
2. **¿Cree conveniente tener la posibilidad de sincronizar sus archivos en el computador con Alfresco al estilo Google Drive?**

Si_____ No_____

3. **¿Cuándo realiza la carga de contenido en Alfresco, existe la posibilidad de subir archivos mediante arrastrar y soltar?**

Si_____ No_____

4. **¿Cree usted que se facilitaría la carga de contenido a Alfresco, si existe la posibilidad de arrastrar y soltar archivos?**

Si_____ No_____

5. **¿Es posible cargar contenido masivamente (muchos archivos a la vez)?**

Si_____ No_____

6. **¿Necesita cargar archivos masivamente?**

Si_____ No_____

7. **¿La búsqueda de contenidos en Alfresco resulta complicada?**

Si_____ No_____ ¿Por qué? _____

8. **¿Sería útil poder agregar metadatos personalizados (características de los contenidos) para efectivizar las búsquedas de contenidos?**

Si_____ No_____

9. **¿Ha tenido algún problema con Alfresco?**

Si_____ No_____ ¿Cuál?

_____

_____

10. **¿Qué mejoras le aplicaría a Alfresco?**

_____

_____

_____

_____

ANEXO II

Código

El código de la aplicación está en el CD adjunto.

ANEXO III

Diagrama UML de clases e interfaces

Por su extensión este ANEXO se muestra en el CD adjunto.

ANEXO IV

Modelo de encuesta para pruebas de usuario

Instrucciones. Marque con X según su criterio.

1. **¿Es posible iniciar sesión en la aplicación con las credenciales de Alfresco?**

Si_____ No_____

2. **¿Es posible navegar entre el Navegador de Repositorio y la búsqueda de Contenidos en la aplicación?**

Si_____ No_____

3. **¿Si se modifica un archivo en el Navegador de Repositorio de la aplicación, el cambio se ve reflejado en Alfresco Share?**

Si_____ No_____

4. **¿Es posible cargar archivos y carpetas masivamente a través de arrastrar y soltar?**

Si_____ No_____

5. **Se ha definido el Tipo alfepn:Prueba con los metadatos alfepn:Metadato_prueba y alfepn:Metadato2_prueba ¿Es posible aplicar esos metadatos a un a un archivo ?**

Si_____ No_____

6. **¿Es posible añadir un valor al metadato alfepn:Metadato2_prueba?**

Si_____ No_____

7. **¿Cuándo se modifica un archivo en el repositorio local de la PC y luego de sincronizar, el cambio se ve reflejado en Alfresco Share?**

Si_____ No_____

8. **¿Cuándo se modifica un archivo en Alfresco Share y luego de sincronizar, el cambio se ve reflejado en el repositorio local de la PC?**

Si_____ No_____

9. **¿Es posible buscar el metadato alfepn:Metadato2_prueba según el título del metadato?**

Si_____ No_____

10. **¿Es posible buscar el metadato alfepn:Metadato2_prueba según el valor del metadato?**

Si_____ No_____

ANEXO V

Manual de usuario

Debido a su extensión este manual de usuario se encuentra en el CD adjunto.

ORDEN DE EMPASTADO