Universidad Tecnológica Nacional
Facultad Regional Buenos Aires

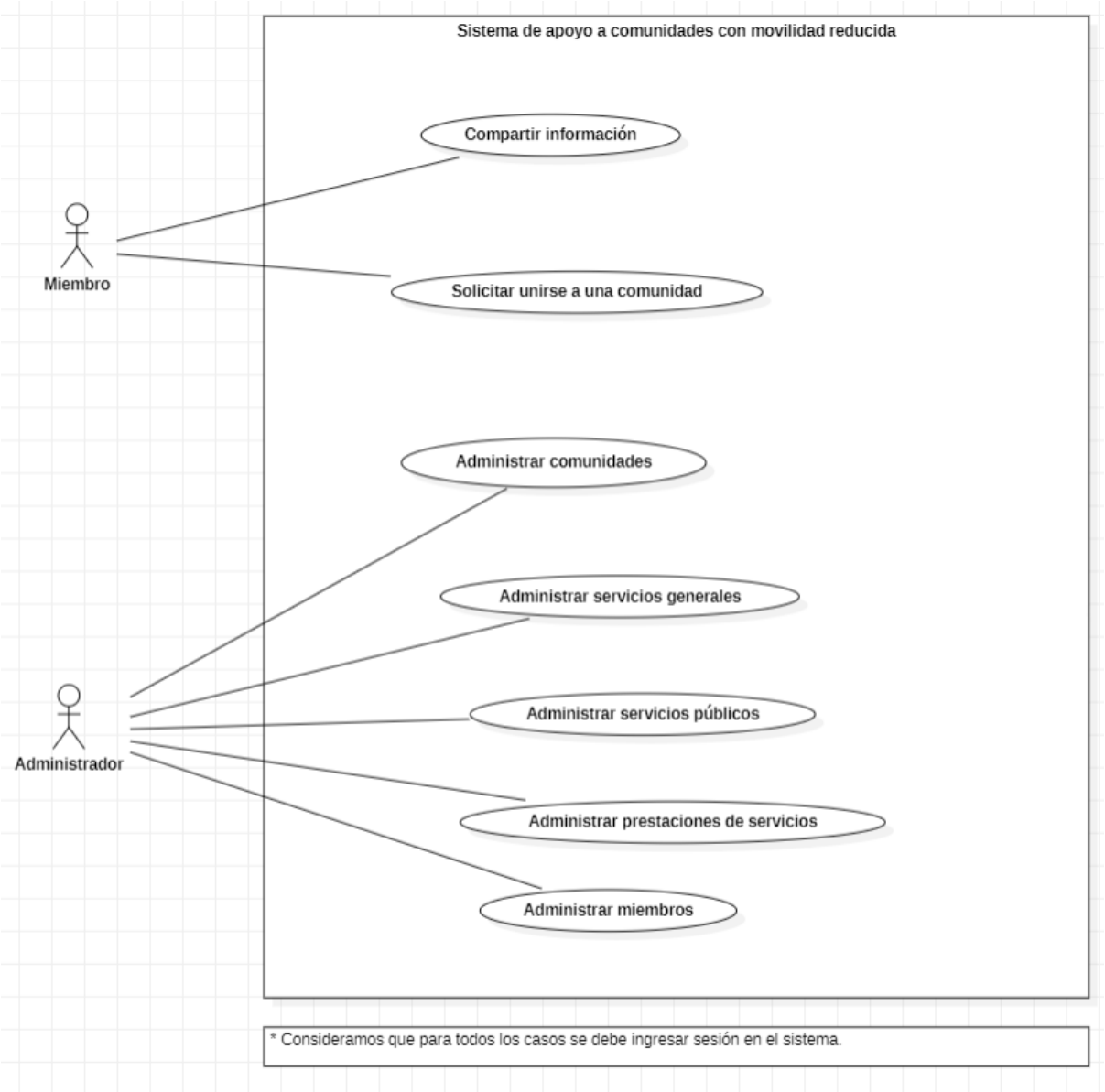
Diseño de Sistemas
Curso: K3001/3101
Turno: Mañana

TP Anual
Sistema de Apoyo a Comunidades con Movilidad Reducida
Primera entrega

GRUPO N° 4	
Nombre y Apellido	**Legajo**
Carriquiry Castro, Joaquin	2026703
Lamas, Chabela María	2027148
Montenegro Aguilar, Gabriel Montenegro	2038742
Pérez Gribnicow, Irina Maia	1781650
Sangroni, Luciano Gabriel	2040440

FECHA DE PRESENTACIÓN:	25/04
FECHA DE DEVOLUCIÓN:	
CALIFICACIÓN:	

1. CASOS DE USO



Para esta primera entrega consideramos dos actores fundamentales: Miembro y Administrador. Decidimos no contar con un actor Proveedor cuyo accionar sea designar a los administradores y validar que no tengan conflictos ya que esta es una funcionalidad que implementará el sistema.

Por un lado, establecimos que un administrador podrá, valga la redundancia, administrar servicios generales, servicios públicos, prestaciones de servicios, comunidades y miembros. Esto se debe a que las acciones previamente mencionadas forman parte de los requerimientos generales del sistema, considerando administración a las acciones de alta, baja y modificación. Además, nos pareció que estas tareas de administración no debían ser ejecutadas por un actor usuario genérico sino uno con mayores privilegios.

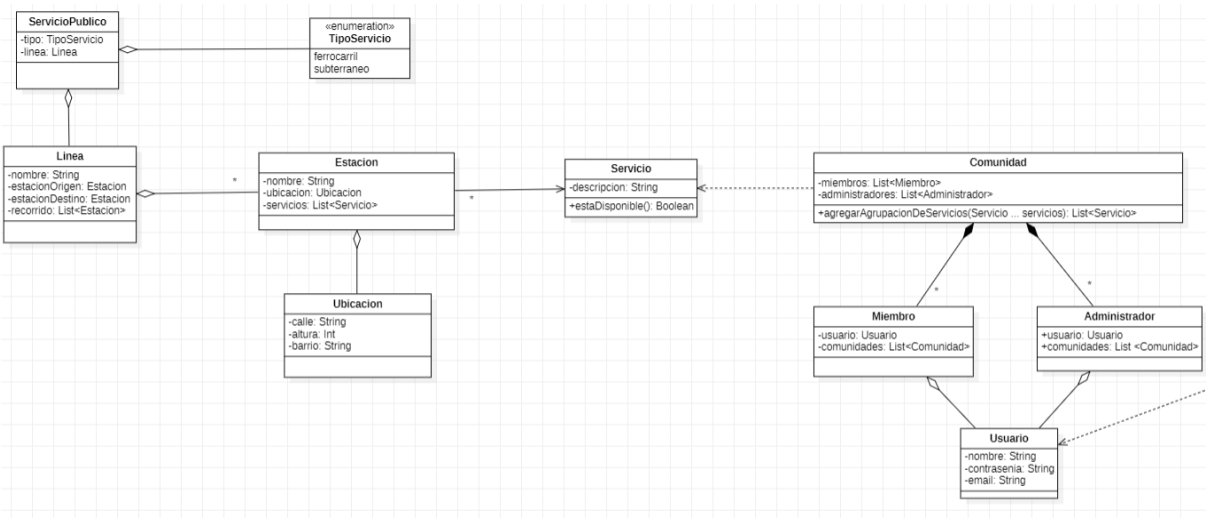
Por otro lado, los miembros son capaces de compartir información, recomendaciones, entre otras cosas, con su comunidad, siendo capaces, de la misma manera, de solicitar unirse a una.

Cabe aclarar que, como se menciona en la sección inferior del presente diagrama, para realizar todas estas acciones, los miembros y administradores, deben previamente ingresar sesión en el sistema (con su nombre y contraseña). De la misma manera, no representamos a los usuarios en este diagrama ya que éste es un acceso al sistema que no debe ser representado como un actor externo.

2. DIAGRAMA DE CLASES

Para lograr una explicación más comprensible, decidimos subdividir el diagrama de clases en dos, en primer lugar la parte de dominio y posteriormente el registro y las validaciones:

A. DOMINIO

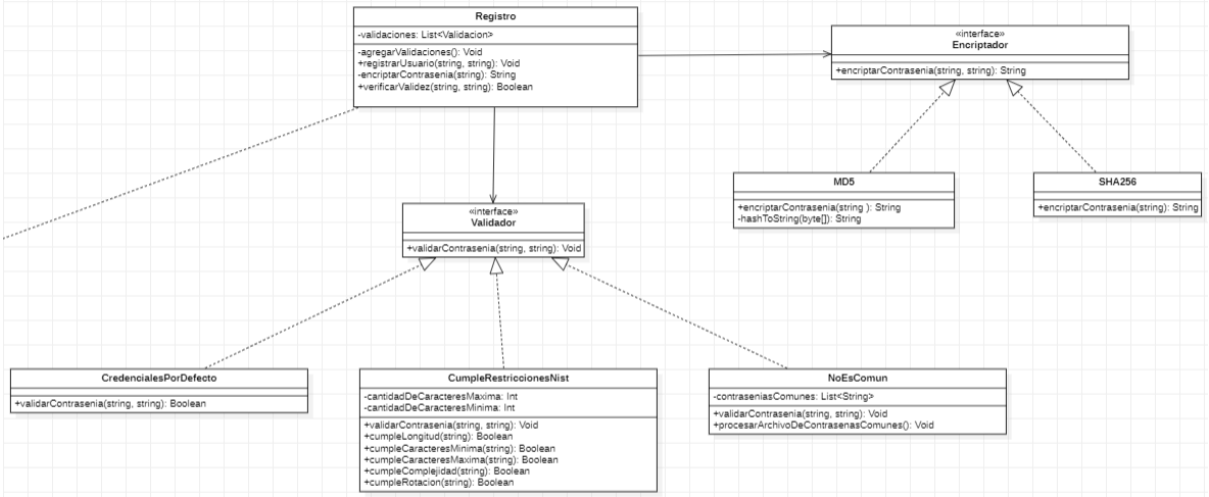


Para comenzar, a partir de la lectura del enunciado, consideramos una clase a *servicio público* ya que nos permite agrupar los tipos de servicios con su correspondiente *línea*. Cabe aclarar que, el *tipo de servicio* lo representamos como una enumeración debido a que, por ahora, son meros datos que no cambian a lo largo de la ejecución del programa (muy seguramente, como varias de las decisiones tomadas, sufran modificaciones a lo largo de las entregas, con la suma de nuevos requisitos). De la misma manera, la *línea* es una clase que logra diferenciar a las líneas de transporte con sus direcciones, una característica particular del entorno del presente trabajo. Para ello, se determinan las direcciones a través de la entidad de *estación*, una clase que cuenta con su nombre, *ubicación* y una lista de *servicios*. Los *servicios* cuentan con una breve descripción, la cual va a ser de gran utilidad en el momento de la agrupación de los mismos (existen agrupaciones estándares y otras que pueden ser definidas por determinadas comunidades). De la misma manera, se establece a los servicios como una clase ya que por ahora sólo se requiere que sepan responder si están disponibles o no, por lo que deben ser una clase independiente. No obstante, no representamos clases para los baños o las escalares por la razón de que, al menos en esta primera entrega, no contienen funcionalidades particulares.

Por otro lado, existen *comunidades* que conocen a sus *miembros* y *administradores*. Como se mencionó anteriormente, éstos tienen la capacidad de agrupar servicios. Los *miembros* y *administradores* pueden pertenecer a más de una *comunidad* y se encuentran

relacionados con un *usuario* en específico. Cabe aclarar que, diferenciamos a los *miembros* y *administradores* con los *usuarios* ya que los dos primeros tienen un acceso al sistema mientras que el otro es un acceso al mismo.

B. VALIDACIONES



Tomamos la decisión de que nuestro sistema contara con *usuarios*, que representarán accesos al sistema, para ello creamos una clase *registro* con un método para instanciar *usuarios*, dicha instanciación se realizará solo en caso de que el nombre y contraseña elegidos pasen por una serie de comprobaciones.

Para realizar las comprobaciones decidimos que el *registro* cuente con una lista de elementos con la interfaz "*validación*" y que solo se instancie un *usuario* si pasa todas las *validaciones*, este modelo nos permitirá agregar más validaciones en caso de ser necesario en un futuro y sobretodo pensamos este modelo imaginando que por medio de excepciones se puede comunicar exactamente por qué un *usuario* no puede ser creado.

Para la clase *no es común* decidimos implementar un algoritmo que permita leer un config que contenga las 10 mil contraseñas más comunes (en caso de no poder leerlo debe lanzar excepción) y compare la contraseña introducida con las del archivo.

Este fue el archivo utilizado para las contraseñas más comunes: https://github.com/OWASP/passfault/blob/master/wordlists/wordlists/10k-worst-passwords.txt

Para la clase *credenciales por defecto* buscamos información al respecto y entendimos que el nombre no debe ser igual a la contraseña basados en estas fuentes:

- https://github.com/OWASP/passfault/blob/master/wordlists/wordlists/10k-worst-passwords.txt

- https://cidecuador.org/wp-content/uploads/congresos/2018/congreso-internacional-de-desarrollo-de-software/diapo/principales-vulnerabilidades-en-aplicaciones-y-como-evitarlas_carlos-gonzales.pdf

En la tercera clase agrupamos todas las características de *Nist* ya que consideramos que van relacionadas a aspectos sintácticos de la contraseña. El documento utilizado es el siguiente: https://pages.nist.gov/800-63-3/sp800-63b.html#memsecret

Por un lado consideramos la extensión de la contraseña con un mínimo de 8 caracteres y un máximo de 64 como establece el documento.

Por otra parte, tuvimos en cuenta la complejidad, donde utilizamos Pattern regex que particularmente en el lenguaje Java permite utilizar expresiones regulares, para así asegurarnos que tenga nuestra contraseña al menos una minúscula, una mayúscula, un carácter especial y un número. Cabe aclarar que el requerimiento de Rotación fue considerado no relevante para esta entrega.

De la misma manera, decidimos implementar un método que pueda encriptar las contraseñas por medio de hashes (permiten convertir los valores de entrada en otros valores de 32 bytes) para mejorar la seguridad del sistema. Para ello, realizamos una interfaz con dos posibles algoritmos (MD5 y SHA-256). Las fuentes utilizadas para realizar lo mencionado anteriormente son:

- https://www.baeldung.com/sha-256-hashing-java
- https://www.baeldung.com/java-md5