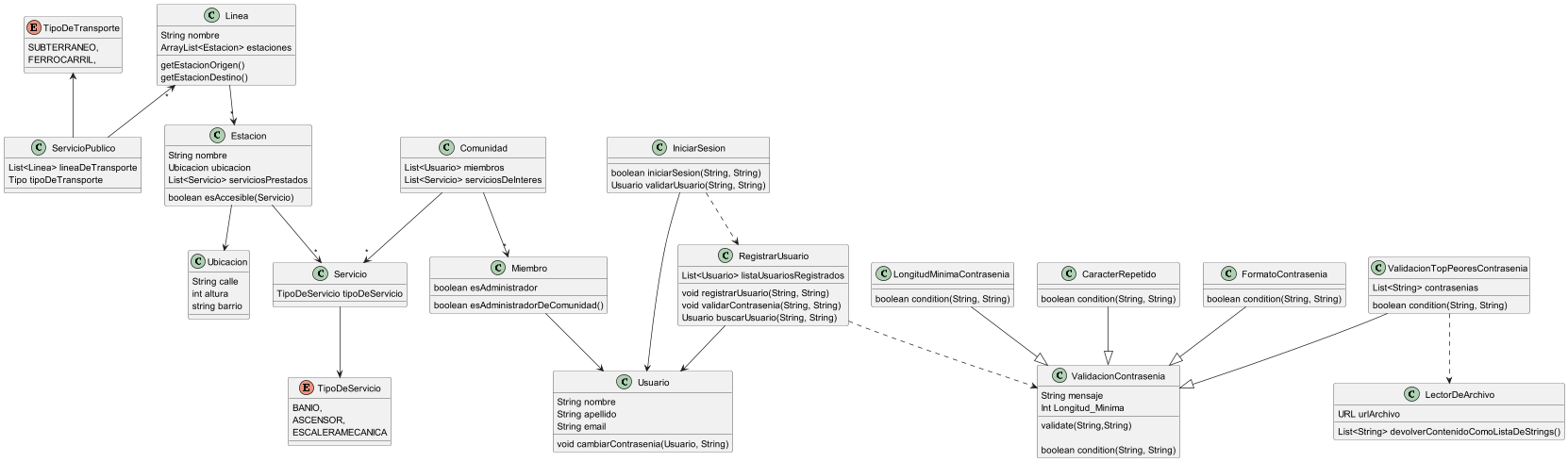
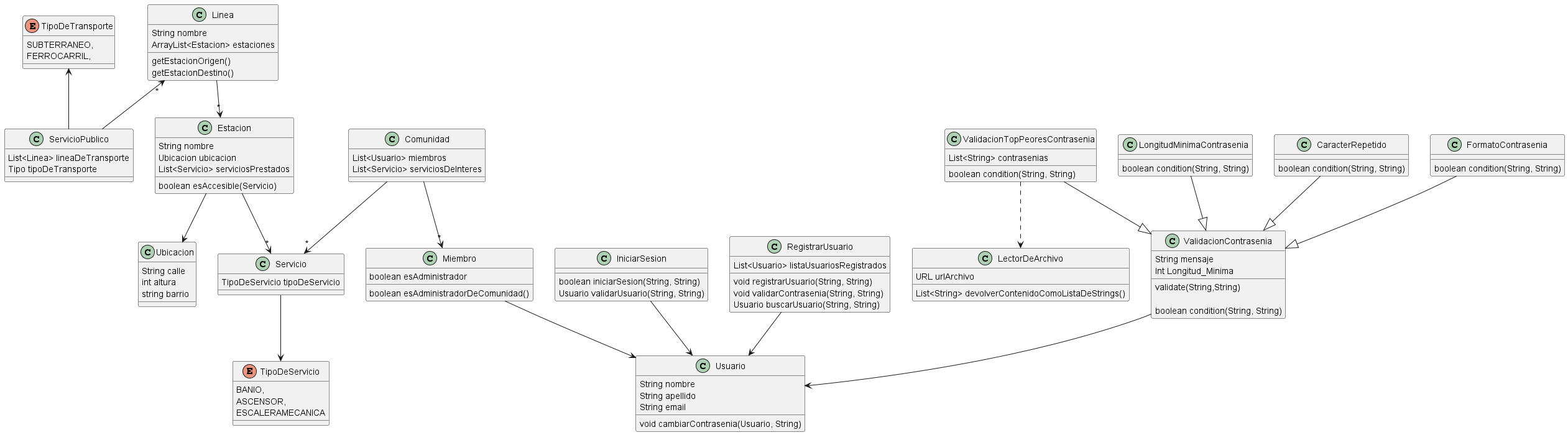
Decisiones de Diseño Primera Iteración.

**Diagrama de clases: **

**Servicio Público**

Como se indica en el enunciado, esta primera iteración se dedica a los servicios públicos de transporte de subterráneos y ferrocarriles. Un servicio público incluye una lista de líneas y el medio de transporte.   
Para esta versión decidimos ir por el lado de la creación de una clase servicio público para generar la abstracción pedida en el requerimiento, esta clase conoce a un enum que contiene los tipos de transporte que se incluyen en esta iteración, decidimos generar un enum, debido a que los tipos de transporte son estáticos en esta versión. Por lo tanto un enum cumple con la funcionalidad y mantiene el sistema simple y robusto (Comparado si el tipo de transporte sea un String que se pueda guardar con el nombre).

Además de esto un ServicioPublico conoce sus líneas, del servicio público.

Una línea conoce una serie de estaciones, el modelado de las Estaciones dentro de la Línea se hizo un un Array de estaciones, ya que creemos que esto es una manera simple de resolver el problema de mantener un orden en las estaciones, siendo la primera estación de la línea la posición 0 del array, y la última estación estación la posición n del array.

**Diagram

Description automatically generated**En cuanto a la estación es una abstracción importante dentro del sistema, es la clase central, ya que es la que tiene conocimiento de los servicios.

La estación conoce además una ubicación, esto es para cumplir con lo que dice el enunciado *“De cada estación se conoce su nombre y su ubicación geográfica”,* nosotros decidimos modelar la ubicación de una estación dependiendo en qué calle está la misma, la altura y en qué barrio está ubicada. Por el momento la calle, altura y barrio son todos atributos dinámicos, que no tienen ningún tipo de validación, esto para simplificar un poco el modelo de datos. Esto es a costo de perder un poco la robustez del sistema.

**Servicios**

**Diagram

Description automatically generated**Para los servicios, lo que priorizamos es simplicidad y no sobrediseñar el sistema con requerimientos que no se especifican su funcionalidad. Lo que consideramos para la clase de servicios es únicamente un identificador de tipoDeServicio que es un enum. Esto es debido a que hay distintos tipos de servicios nombrados en el enunciado “*servicios que el sistema deberá considerar son: escaleras mecánicas, ascensores y baños”*.

Consideramos que además el requerimiento de agrupación no cumple ninguna funcionalidad, por lo tanto no tiene sentido modelar una clase que lo único que haga es tener una lista de servicios. Es una abstracción interesante de tener pero no tiene mucho sentido en cuanto a la funcionalidad.

**Diagram

Description automatically generatedPrestación de Servicios**

Para la prestación de servicios lo que pensamos es que la clase estación conozca a varios Servicios, entonces modelamos una lista de servicios dentro de la estación. En esta lista están todos los servicios disponibles para esta estación. Los servicios que no se encuentran en esta lista son servicios no prestados. Esto nos da simplicidad para la resolución de de un futuro algoritmo para calcular la accesibilidad de una estación frente a un grupo determinado de servicios. Ya que preguntamos si existe en la lista de la estación todos los servicios del conjunto.

**Diagram

Description automatically generatedComunidad y miembros**

En cuanto a la comunidad, optamos por hacer una clase que conozca una lista de miembros y una de servicios, manteniendo la cualidad de simplicidad. Para los miembros decidimos hacer una clase aparte que represente el rol que cumple en cada comunidad, lo que nos permite flexibilidad. De esta manera, un usuario puede ser administrador y miembro para distintas comunidades o cambiar de rol con mayor facilidad.

**Usuario de la plataforma**

Cada uno de los componentes que participan en la seguridad mantienen su cohesión. Debido a esto, priorizamos la cualidad del desacoplamiento ya que cada clase tiene sus propias responsabilidades.

**Usuario**

Para la clase Usuario tomamos la decisión de buscarle simplicidad y optamos a que conozca a las clases Miembro, IniciarSesion y RegistrarUsuario.

**Registración**

Para la clase registración decidimos que conozca a validacionContrasenia y así poder modelar una lista de validaciones, que tiene que cumplir la contraseña antes de poder registrar el usuario. De vadalicionContrasenia se heredan las clases ValidacionTopPeoresContrasenia, LogintudMinimaContrasenia, CaracterRepetido y FormatoContrasenia, que son las encargadas de las validaciones de la contraseña ingresada. De esta forma podemos validar de forma independiente la contraseña ingresada y en caso de algún error poder dar un mensaje más preciso y personalizado por nosotros.

**Iniciar Sesión**

Es una clase que es conocida por el usuario, decidimos crear esta clase para poder separar la funcionalidad y poder tener flexibilidad a futuros cambios.