Usuario

Se decidió que la clase **AdministradorUsuario** sea la que realiza las acciones generales de cada usuario (registro, cambio de contraseña y el inicio de sesión). Al registrar un nuevo usuario, utiliza al ValidadorUsuario para validar la contraseña según las necesidades que el sistema requiera.

La clase **ValidadorUsuario** tiene una lista de estrategias del tipo de la interface **EstrategiaValidacion** que, en forma de patrón Strategy, permite incorporar nuevas estrategias a futuro. Con el sistema, se desarrollan las estrategias de **RegExp** y de **NoEstaEnLista**. El método esValido(usuario, password) utiliza validarPorUsuario y validarPassword. El primero de esos métodos verifica que el username sea distinto a la password, y el segundo recorre la lista de estrategias.

- **EstrategiaValidacionRegExp** compara el string recibido por parámetro (password) con una expresión regular. Aquí se verificaría, por ejemplo, el largo de la password o que contenga ciertos caracteres (mayúsculas, minúsculas, símbolos, números, etc) según el sistema lo requiera.
- Por otro lado, **EstrategiaValidacionNoEstaEnLista** tiene un atributo del tipo **ObtenerListaString**. Esta interface que, en forma de Adapter, permite obtener una lista de String para que EstrategiaValidacionNoEstaEnLista lo compare con la cadena (password) y verificar que esa cadena no se encuentre en esa lista obtenida. **ObtenerPeoresPasswordsURL** es una de esas formas adaptada de obtener la lista.

Al haber una sesión por usuario, decidimos incluir un objeto de clase **Sesion** que encapsule la lógica del delay ante repetidos intentos fallidos de inicio de sesión (como se indica en los requerimientos de seguridad).

Con respecto al hasheo, existen varios algoritmos y si bien algunos sólo necesitan de una clave original para devolver una hasheada, existen otros que utilizan datos que es necesario guardarlos de manera individual (están fuertemente relacionados con una clave original en concreto). Tal es el caso de la **salt**, que es utilizada por el algoritmo **PDBFK2**. Es por esto que decidimos crear la clase **Password**, la cual no sólamente guardará la clave hasheada, si no que contendrá una instancia de **EstrategiaHash**, la cual se encargará de la lógica de hasheo y opcionalmente guardará datos relacionados a la clave original (dependiendo de la estrategia concreta). Ninguna entidad usa **EstrategiaHash** porque no queremos que el tipo de hasheo varíe en tiempo de ejecución, sino que buscamos que la flexibilidad del cambio se dé desde el código mismo.

Además, cada usuario tendrá asignado un **RolPlataforma**. Esta interface es implementada por **AdministradorPlataforma** y **UsuarioPlataforma**. Decidimos

incluir estas clases en el diagrama aunque por el momento no tengan comportamiento definido, porque consideramos que a futuro tendrán mucha relevancia para determinar las funciones de cada tipo de usuario en la plataforma.

Comunidad

Además, cada usuario tendrá una lista de membresías. Cada **Membresia** tendrá referencia a su Usuario y a la **Comunidad** a la que pertenece ese usuario. Por ese motivo, se tendrá una Membresia por cada comunidad a la que pertenece un usuario. La Membresia, a su vez, tiene un atributo del tipo **RolComunidad**.

La interface **RolComunidad** es implementada por los roles que puede tener un usuario en una comunidad (**RolAdminComunidad**, **RolAfectadoComunidad**, **RolObservadorComunidad**). Se decidió implementar el rol en Membresía para permitir que un miembro pueda cambiar de rol en tiempo de ejecución. Las acciones que pueda realizar un usuario en una comunidad dependerán de su rol. Por ejemplo, si un usuario desea ver su menú de opciones dentro de una comunidad, si es administrador probablemente tenga más opciones que si es solo afectado.

Como las comunidades deben definir servicios, estas guardan la lista de **Servicios** creados entre sus atributos.

Servicio

Para modelar los distintos servicios posibles, hemos decidido implementar el uso de **Etiquetas**, las cuales podrán ser creadas y modificadas por los distintos miembros de las comunidades (como lo indica el enunciado). Por eso, se agrega el método *definirServicio(servicio)* en **RolComunidad** que permite que cada usuario defina un servicio en una comunidad. De esta forma podríamos también agrupar servicios en uno sólo (un servicio con varias etiquetas). Por ejemplo, en el caso de los baños podríamos agruparlos con las siguientes etiquetas:
[“Baño”,”Hombre”,”Mujer”]

Por tema de facilidad para la manipulación de la ubicación de una estacion, se decidió utilizar una clase concreta **Ubicacion** para cierto fin. Por otro lado, una **Linea** guarda en su tipo de línea un valor del enum **TipoLinea** (subte, tren). Se decidió utilizar un enum porque es simplemente para identificar a qué tipo de transporte pertenece, sin ningún otro comportamiento particular.