ENTREGA 1:
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.

ENTREGA 2:
Entidad, Establecimiento y Organismos de Control

La clase **Linea** de la entrega anterior fue reemplazada por una nueva clase **Entidad**.
Los organismos de control y las entidades prestadoras se diseñaron en una única clase denominada **ControlEntidades**. Esta clase tiene un atributo *informado* del tipo **Persona** y un *generadorInformación* de tipo **InformacionAdapter**. Esta interfaz será implementada en próximas entregas con la forma de obtener la información para los informados.
ControlEntidades tiene una lista de entidades que controla/administra según sea organismo de control o entidad prestadora correspondientemente.
Tanto Entidad como **ControlEntidades** tienen un atributo del tipo **Denominación** que permite especificar una descripción adicional, esto es útil para próximas entregas, según indica la consigna.

Ubicación

Originalmente, la clase **Ubicación** tenía sólo dos atributos: *latitud* y *longitud*. Con la nueva necesidad de identificar a las ubicaciones por provincia, municipio y/o departamento, se agregó un nuevo atributo: *localizaciones*. Esta lista es de tipo **Localizacion,** y contiene el listado de localizaciones atribuibles a esa ubicación. Al optar por tener una lista en lugar de atributos separados podemos buscar dentro de *localizaciones* cualquier localización de cualquier tipo y eventualmente, si fuera necesario agregar un nuevo valor en **TipoLocalización,** dicho cambio no impactaría en Ubicación.

La clase **Localizacion** representa tanto a provincias como a municipios y departamentos. Dicha clase consta de los atributos: *Id (***String***), nombre (***String***)* y *tipo (***TipoLocalización***).*
TipoLocalizacion, es un enum que detalla si dicha localización es Provincia, Municipio o Departamento. Este enum especifica formalmente el tipo para este dominio, esto lo hicimos así para evitar acoplarnos a las convenciones de la api GeoRef que segmentan las localizaciones y las diferencian mediante el *id*.
Utilizamos **Localización** tanto para asignar localizaciones de interés para una instancia **Persona** así como también para especificar datos de localización de una **Ubicación**.

Etiqueta

Se decidió cambiar los atributos de esta clase, dejando el id y agregando un tipo y un valor. Permitiendo reutilizar etiquetas de forma más sencilla.

Interés

Se decidió diseñar al interés como una clase aparte, que es referenciada por una Persona. Este interés tiene la lista de entidades prestadoras, la lista de servicios y la localización de la persona.
La forma de encontrar los servicios sin disponibilidad en función del interés de la persona sería: a partir de la lista de **EntidadesPrestadoras** de interés, realizar un primer filtrado obteniendo los Establecimientos de la locación de interés. Luego, se obtienen los **servicios**

no disponibles de dichos establecimientos. Finalmente, se compara esa lista de servicios no disponibles con los **servicios de interés** de la persona.

ImportarEntidadAdapter e ImportarEntidadCSV

Se definió la interfaz **ImportarEntidadAdapter** que tiene la firma del método **importar(path:String) : List<ControlEntidades>**, este método sirve para obtener un listado de ControlEntidades en base a un path de un archivo csv. Es decir que dicho método sirve para importar “empresas prestadoras de servicios” y también “organismos de control”, además de las entidades.
Además se define la clase ImportarEntidadCSV que implementa **ImportarEntidadAdapter** e internamente tiene la lógica para procesar un CSV y devolver un listado de ControlEntidades.

GeoRef API

Para comunicarnos con esta API REST decidimos diseñar la clase concreta **ServicioGeoRef** que, utilizando Retrofit y la interfaz **GeoRefService**, retorna los listados de provincias, municipios o departamentos. Además, retorna un objeto **UbicacionGeoRef** según la latitud y longitud dadas. También, **ServicioGeoRef** tiene métodos privados para generar los listados de Localizaciones que se retornarán al controlador. De esta forma, se normalizan los datos con la clase **Localizacion**, que es usada en el dominio del sistema.