Documentación TPA - 1era entrega

Decisiones de diseño tomadas y su justificación:

Como primera aproximación decidimos modelar la clase **Servicios Públicos** en la que podemos establecer las líneas existentes y el tipo de transporte del servicio en cuestión, a su vez el sistema deberá permitirnos agregar, eliminar y modificar el estado en que se encuentran las líneas de los diferentes servicios

Luego preferimos modelar la clase **Línea** (ya que un servicio puede poseer muchas líneas), las mismas deben poseer un nombre que las identifique así como también su estación de origen, destino, el conjunto de estaciones que componen a la misma y el estado. También se deben poder agregar y eliminar estaciones a la línea.

Decidimos hacer un enum para los **Estados** ya que es una cantidad de estados finita la cual no se debe modificar frecuentemente, y además para poder cambiar entre estados fácilmente como se requiere para Servicio y Línea.

Elegimos crear la clase **Comunidad** con atributos descripción, la cual incluye de qué trata la comunidad en cuestión y cualquier detalle que los administradores deseen agregar, una lista de los miembros y una lista de los administradores. Además cuenta con atributos para administrar los miembros, saber si un miembro determinado es administrador, y agregar un servicio a una determinada estación, función clave que deben poder cumplir todas las comunidades.

La clase de **Miembro** contiene una lista con las comunidades a las que pertenece, y métodos para unirse, o irse de comunidades específicas, además de otro método que le permite saber si es administrador de alguna comunidad en particular.

También creamos la clase **Servicio** a partir de la cual se podrá definir el estado del mismo y una descripción en la que se podrá aclarar si el servicio es un baño, que contenga al de mujeres, hombres, discapacitados, etc.

Generamos además una clase **Estación** la cual cuenta con un nombre, su ubicación y los servicios que posee. En la misma podremos agregar, eliminar o modificar el estado de cada servicio que contenga.

Por último, la **Ubicación**, que está compuesta por una latitud y una longitud, decidimos modelar de esta forma para garantizar la unicidad de la ubicación de cada estación.

En lo que respecta a los **requerimientos de seguridad**, decidimos guiarnos por un patrón strategy, ya que nos permite mantener un conjunto de algoritmos o requerimientos para conformar una contraseña y que los mismos sean modulares y extensibles respecto de las necesidades del cliente.

Documentación TPA - 2da entrega

02/05: Comenzamos a modelar el diagrama de objetos. Tenemos dudas con la clase entidad, ya que pensamos hacerlo con una interfaz, pero solamente lo estaríamos haciendo para 2 atributos de Línea, por lo que no sabemos si es del todo conveniente. También tenemos dudas sobre si los intereses del usuario deberían ser Entidades o Establecimientos, ya que al usuario le debería interesar un Establecimiento específico de una Entidad basado en su localización, pero no toda la Entidad en sí.

Cambiamos la clase Línea por una llamada Entidad, y la clase Estación por una llamada Establecimiento, las cuales mantienen comportamiento muy similar a las anteriores.

Además agregamos un Enum de Tipos de Servicio, para que el usuario pueda ingresar los servicios asociados manteniendo consistencia de datos.

Por último, cambiamos el Enum Transporte por uno llamado TipoEntidad, en el cual están listados todos los tipos posibles de entidad, con el fin de nuevamente mantener la consistencia de datos.

24/05: Agregamos la clase Localización la cual utiliza la API REST Georef para que cada entidad y cada miembro tenga una asociada para modelar la localización de interés. En Entidad y Miembro ya no hay más una lista de localizaciones, sino que hay un solo objeto de la clase Localización. Quitamos el estado DEMORADO de el enumeration Estado. Agregamos 2 métodos a la clase Entidad llamados tieneAtributosExtra e intanciarAtributosExtra, para intentar de solucionar el problema de que solo algunas entidades tendrían estación de origen y destino. Agregamos el metodo serviciosDeInteres a la clase Miembro. Actualizamos el diagrama de casos de uso para que contemple las nuevas decisiones de la entrega.

01/06: Comenzamos con el metodo serviciosDeInteres del miembro, el cual no logramos terminar de pensar la lógica, ya que no sabemos donde hacer que se agreguen los servicios a la lista, porque pensamos que se estaría rompiendo el encapsulamiento si hacemos que se agreguen desde los servicios, y es muy complicado traer los servicios con problemas para agregarlos desde el miembro.

02/06: Realizamos el metodo serviciosDeInteres del miembro, el cual nos trajo muchos problemas. Lo que hicimos fue filtrar todas las entidades por las que le interesan al usuario, luego dentro de cada una de esas entidades, filtramos dentro de cada uno de los establecimientos de ellas para buscar los servicios que le interesan al miembro, y filtrar por los que tiene problemas. Traemos los servicios con problemas y los agregamos a una lista anteriormente creada. Al final de todo, filtramos los servicios que quedan para eliminar los repetidos. Un suceso importante fue la realización de que no era importante establecer el origen y el final de una posible entidad subte o tren, sino que más bien era importante en el diseño de la entrega pasada en el cual se daba importancia a la línea en sí.

05/06: Terminamos con la implementación de la API. Nos basamos en los ejemplos planteados en el seminario para agregar las partes de los departamentos. Actualizamos todos los diagramas para que cumplan con las implementaciones realizadas en el código.