BITÁCORA TP ANUAL: GRUPO 9

Integrantes:

- **Dellafiore, Matias**
- **Hernandez Canetti, Diego**
- **Martina Lopez, Avril**
- **Navarro, Ramiro**
- **Paoli, Juan Francisco**

ÍNDICE:

Primera Entrega - 25/04

CASOS DE USO:

- En esta primera entrega, no se hace hincapié en la diferencia en las funcionalidades, dentro del sistema, entre los usuarios de la plataforma, usuarios administradores[1] y miembros de la comunidad, por lo tanto, se graficó como **usuario** con las funcionalidades básicas explicitadas en los requerimientos. En un futuro esto puede cambiar y se implementarán las modificaciones necesarias.
- El ente regulador de servicios consideramos que al ser un organismo estatal sus actividades ocurren por fuera del sistema. En cuanto al reglamento del estado de los servicios dentro del sistema, consideramos que es el propio sistema quien se encarga de ello.

DIAGRAMA DE CLASES:

MÓDULO SERVICIOS Y USUARIO

CLASE SERVICIO PÚBLICO

Decidimos modelar como una simple clase para esta entrega, ya que no conocemos el comportamiento que tendrán estos servicios, ni tampoco si todos tendrán el mismo comportamiento o no. Por lo tanto, se utilizó un atributo tipoDeServicioPublico de tipo enumerado para identificar el servicio que se quiere inicializar(por el momento, todos tendrán el mismo comportamiento).

INTERFAZ SERVICIOS

En la consigna quedó especificado que cada uno de los servicios va a tener un mismo comportamiento, pero estos van a tratar los métodos de manera diferente.

La última consideración, es que en un futuro, podremos agregar más servicios de manera polimórfica a nuestro sistema, sin necesidad de romper las estructuras ya predispuestas.

SERVICIOS DE ELEVACIÓN

Se usa un enumerado para los distintos tipos de servicios de elevación: ascensores y escaleras mecánicas. Decidimos hacer esto ya que a futuras entregas se pueden ir

[1] Asimismo, no se integró la idea de conflictos de intereses ya que esto va por afuera del sistema

agregando más servicios de elevación y por el momento no hay ninguna diferencia entre la funcionalidad de cada uno dentro del sistema (todos usan los mismo métodos).

Para el tramo, se optó por una solución donde el tramo no sea una clase con métodos, esto es porque el enunciado nunca menciona: de que se tenga que calcular el tramo, que el tramo afecte servicio, que tenga básicamente algún comportamiento. Por lo anteriormente mencionado, decidimos diagramarlo como una clase de tipo Enum que diga qué parte se está recorriendo, y de esta manera lograr una mejor abstracción.

PERSONA Y USUARIO

En nuestro diseño hemos decidido considerar una persona (física) y un usuario como la misma entidad, en donde el usuario toma los atributos de la persona que lo crea. El motivo de esta decisión es que en el sistema los que interactúan con los servicios son usuarios de la plataforma, no “personas”.

CLASE USUARIO

En el presente diseño no se tomó la clase usuario como una interfaz o una abstracción ya que, por lo mencionado anteriormente, no se han hecho diferenciaciones en las funcionalidades de los diferentes tipos de usuarios. Esto se puede implementar a futuro de ser necesario ya que nuestro diseño es adaptable.

REPOSITORIO DE USUARIOS

Se agregó una clase llamada repositorio de usuarios para el manejo de los mismos a través de una lista con todos los usuarios que se van creando en el sistema. Además, esta clase se encarga de validar los nombres de usuario, es decir, controla que no exista un usuario ya registrado con el mismo usuario (en caso de ser así, lanza una excepción evitando que el usuario pueda registrarse con dicho usuario).

MÉTODO REGISTRAR USUARIO

El método es void en lugar de bool, ya que no es necesario preguntarle al validador ed contraseñas, si la contraseña es válida o no, sino que es más eficiente decirle que la valide, y, en caso de ser inválida, cortar la ejecución con una excepción y un mensaje indicando que requisitos de seguridad incumple. Por otro lado, si se valida la contraseña, simplemente continúa ejecutándose el programa.

SERVICIOS PÚBLICOS Y LÍNEAS:

Consideramos que los ferrocarriles y los subterráneos tienen como atributo la línea que recorren. Una vez que completan su viaje, otra instancia es la que realiza el tramo de vuelta siguiendo una nueva línea (en la dirección contraria).

MÓDULO SEGURIDAD

Hemos decidido trabajar toda la parte de Seguridad del Sistema tiene un comportamiento totalmente diferente al de Usuario y Servicios. La modularidad ayuda además a la mantenibilidad del sistema a futuro.

CLASE VALIDADOR_CONTRASENIAS

Esta clase implementa una serie de reglas que toda contraseña debe cumplir para poder ser validada. Esto proporcionará:

- Mayor facilidad a la hora de agregar nuevas funcionalidades
- Mayor facilidad para corregir comportamiento ya integrado
- Mayor facilidad a la hora de entender el código(y el diagrama)
- Mayor facilidad para eliminar comportamientos obsoletos o ya no usados.

CONTRASEÑA

- Las contraseñas se van a almacenar como un string, y no como una clase, ya que estas no presentan inconsistencia de datos. Además, no consideramos necesario contar con una clase contraseña, ya que no necesitan atributos ni métodos propios.
- Tienen integradas todos los requerimientos pedidos para esta entrega, además de algunos extra para aumentar la seguridad de los usuarios dentro de la plataforma(en caso de que se quiera modificar esto, por la manera en que fue diseñado el código, es muy sencillo eliminarlo).
- Las especificaciones que deben cumplir las contraseñas se encuentran delegadas en clases que implementan una misma interfaz “ReglaContrasenia”. De esta manera, nos aseguramos todas tengan el mismo comportamiento, pero implementado a su determinada manera. Algunas de las reglas necesitan comparar la entrada con unos archivos en busca de símbolos, palabras de diccionario y contraseñas comunes. Para ello, esta información se almacena en archivos.txt, los cuales se encuentran en una configuración a parte. Esto permite una mayor facilidad a la hora de modificar y/o realizar un mantenimiento de estas teniendo en cuenta los requerimientos del usuario.

EXCEPCIÓN REGISTRO DE USUARIO

Para lograr una ejecución del programa más fluida y fácil de entender para los usuarios. En casos en los que la contraseña/nombre de usuario que quisieron utilizar no fuese válida (según los requerimientos definidos para este sistema), la excepción a lanzar permite que estos sepan que fue un error durante su registro, y deberán realizar uno o más cambios para lograr registrarse. Debido a que no consideramos necesario “catchear” la excepción, esta clase hereda de RuntimeException

Segunda Entrega - 07/06

Clase Abstracta Entidad:

Se ha diseñado esta clase como abstracta para permitir la creación de entidades con leyendas apropiadas según su tipo. Por lo tanto, existen dos clases hijas: EntidadDeEstablecimiento y EntidadDeTransporte. Además, las entidades de transporte pueden ser ferrocarriles o subterráneos, por lo que tienen un atributo adicional para identificar su tipo.

Clase MiembroDeComunidad:

El sistema considera cualquier persona que sea usuario del mismo (se grafica). Esta clase representa a las personas que utilizarán el sistema y designarán servicios/entidades de su interés, que se encuentren dentro de su localización, sobre las cuales les gustaría recibir notificaciones. En base a estos intereses van a poder unirse a aquellas comunidades que compartan intereses en común. Estos usuarios deben crear una cuenta con una contraseña para acceder a las funcionalidades del sistema.

Clase Comunidad:

Esta clase representa un conjunto de miembros que comparten un mismo interés. En un futuro va a poder implementar servicios, en base a las necesidades e intereses de sus miembros.

Clase Abstracta Empresa:

Se ha diseñado esta clase como abstracta para permitir la creación de empresas con leyendas apropiadas según su tipo. Por lo tanto, existen dos clases hijas: EntidadPrestadora y OrganismoDeControl. Se tomó la decisión de diagramar la Entidad y la EntidadPrestadora como dos clases diferentes porque es importante para el dominio (y los requerimientos funcionales) tener la idea de Entidad Prestadora como una empresa que se encarga de asignar personas y mandar información, métodos que no realizan las demás entidades. Otro motivo es la necesidad de comunicar la idea de que los Organismos de Control conocen a las Entidades Prestadoras. Además los métodos que comparten ambas, tienen la misma lógica.

Localización:

- ➔ Uso de interfaz: La localización se ha definido como una interfaz para abarcar las tres posibilidades en el sistema: provincia, departamento y municipio, de manera indistinta. Esto significa que un usuario/entidad/establecimiento puede tener definida su localización como provincia, departamento o municipio.

- ➔ Localización del usuario:En el futuro, los interesados en los servicios serán responsables de definir sus localizaciones de interés (de las cuales esperan recibir información) a través de la interfaz gráfica. Por lo tanto, tienen un método para seleccionar su localización proporcionando un ID. Esta localización puede ser una provincia, un departamento o un municipio, según las preferencias del usuario. Además, el usuario puede definir una o varias localizaciones, como el hogar y el lugar de trabajo.

- ➔ Localización de entidades y establecimientos: Al igual que los interesados en los servicios, las entidades y los establecimientos deben definir una localización donde se encuentran instalados. En el caso de las entidades, se utiliza una localización de carácter general, probablemente una provincia, donde brindan sus servicios. Por otro lado, los establecimientos definen una localización más específica que se encuentra dentro de la localización definida por la entidad, proporcionando más detalles (departamento o municipio).

- ➔ Departamentos y Municipios: Originalmente, teníamos la intención de trabajar con instancias de municipios que conocieran a qué departamento pertenecen, de la misma manera que cada departamento conoce a qué provincia pertenece. Sin embargo, debido a las limitaciones de la API (georef), solo podemos obtener la provincia a la que pertenece cada municipio y no el departamento correspondiente. Por lo tanto, hemos trabajado con todos los municipios de la provincia a la que pertenecen, sin la capacidad de restringirse a los municipios dentro de cada departamento.

- ➔ Clase Localizador: Dado que entidades, establecimientos y usuarios del sistema definen sus localizaciones de la misma manera utilizando un ID, hemos creado una clase separada para evitar la repetición de código. Esta clase tiene un método que, a partir de un ID, devuelve una localización, ya sea una provincia, un departamento o un municipio. En busca de mejorar la performance del sistema, el localizador primero analiza si el ID es una provincia o un departamento, de esta manera, si es posible se evita analizar si dicho ID es de un municipio. Esto se hizo de esta manera, debido a que en el país hay muchísimos más municipios que departamentos, por ende, a menos que se esté completamente seguro de que sea un municipio, habría que evitar recorrer el listado de dicha clase.

Carga de datos masiva mediante archivo CSV:

Estuvimos analizando cómo atacar el tema de las empresas (organismos de control y entidades prestadoras). Vimos que los archivos CSV permiten incorporar listas mediante comillas dobles, por ende surge la duda de si hacer una super lista con listas individuales para cada organismo de control y su información (es decir, que entidades de control administra, y a su vez toda la información de cada entidad prestadora), o una serie de entradas en el archivo que no utilizaran listas, es decir, cada línea tendrá información de instancias únicas de organismos de control, de las entidades prestadoras, de las entidades, de establecimientos y de servicios. Por ende, para agregar alguna instancia, por ejemplo, un nuevo establecimiento para dicha entidad, o una nueva entidad prestadora administrada por dicho organismo de control, se necesitaría una nueva entrada. Esto consideramos que trae

como ventaja que todas las líneas del archivo mantienen el mismo formato, por lo tanto la lectura se realiza siempre de la misma manera. El inconveniente es que previo a crear una nueva instancia de alguna de las clases, se debe preguntar si esta no fue creada previamente. En cuyo caso, deberá actualizar la información dentro de dicha clase, por ejemplo, agregando un nuevo establecimiento con sus servicios dentro de una entidad ya existente, la cual pertenece a una entidad prestadora y está a un organismo de control existente.

Diseño archivo CSV:

OrganismoDeControl_nombre, EntidadPrestadoraDeServicios_nombre, Entidad_nombre, ID_Localizacion_Entidad,Establecimiento_nombre,ID_Localizacion_Estalecimiento, Tipo_de_Establecimiento, Servicio, Tipo_de_Servicio,Tipo_del_Tipo_de_Servicio

API:

Hemos establecido un límite máximo de 2500 registros a buscar, ya que en algunas solicitudes necesitamos obtener todos los municipios del país, que según la API son 1.814 (más que la cantidad de provincias y departamentos en el país). Esto nos permite evitar problemas al establecer un número mayor para garantizar la obtención de todos los municipios.

Servicios y Establecimientos:

Los establecimientos son responsables de dar a conocer sus servicios. Es decir, estos servicios se crean como instancias a partir del archivo CSV y se almacenan en un atributo (lista) dentro del establecimiento correspondiente. Por lo tanto, los servicios dependen de los establecimientos. Si un establecimiento deja de existir en el sistema, los servicios asociados a él también deben eliminarse.

Además, hemos considerado que un establecimiento puede tener varias instancias del mismo servicio. Por ejemplo, un establecimiento podría tener 5 baños, y de esos, 2 podrían ser exclusivamente para hombres.

Clase Interés:

Se necesitó la creación de una clase Interés por varios motivos:
1) Es de importancia para TODO el dominio la idea de Interés, va a afectar como funciona nuestro sistema.
2)Una persona tiene puede varios Intereses únicos, por lo tanto, era necesario crear una clase que se pueda instanciar cada vez que se agregue un Interés a su propia lista.

El interés es la intersección de Entidades (y los Establecimientos de estos), Servicios(que se obtienen de los Establecimientos) -> Se puede decir que el interés es un **Value Object.** Esto se puede explicar con un ejemplo, a mi me interesa del banco Santander Río Argentina, sucursal Almagro, sus baños. Dentro del diseño de la clase interés no se guardó el valor de localización ya que se repetiría con el nombre de la Sucursal.