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 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.

LOCALIZACIÓN:

- 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, almacenan localizaciones, sea una provincia o un municipio, las cuales le llegan por parámetro.

- Localización de establecimientos: Al igual que los interesados en los servicios, los establecimientos deben definir una localización donde se encuentran instalados. Esta localización se trata de un municipio.

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.
Por otro lado, cabe destacar el uso del patrón adapter para el desacoplamiento de nuestro código con la librería de manejo de archivos CSV utilizada.

Diseño archivo CSV:
organismoDeControl,entidadPrestadora,entidad,tipoEntidad,establecimiento,nombreLocalizacionEstablecimiento,tipoEstablecimiento,tipoServicio,tipoTipoServicio

Ejemplo: *BCRA, Banco Galicia, Banco Galicia Capital Federal, ESTABLECIMIENTO, Sucursal Recoleta, Capital Federal, SUCURSAL, Banio, DAMAS*

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.
Se utiliza un patrón adapter para desacoplar nuestro código de Georef API. Además, es recibido por parámetro, por lo que es fácilmente mockeable en un test.

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.

Tercera Entrega - 11/07

Incidentes y Reporte de Incidentes:

Se decidió modelar el incidente, conteniendo información sobre el establecimiento y el servicio, y con una lista de reportes de apertura (que permite que se reporte el incidente varias veces) y un reporte de cierre (para que solo sea reportado una única vez el cierre).

Geolocalización:

La geolocalización, o como se llamo en el trabajo, posición, de cada miembro, se definirá a partir de 2 valores, latitud y longitud.
Luego, para calcular la distancia entre 2 posiciones, simplemente se realizan algunos cálculos matemáticos que permiten obtenerla.

Notificaciones:

- ➔ Se utilizó un adapter, para abstraerse de la implementación de las bibliotecas a la hora de enviar notificaciones. En cuanto a whatsapp, se mockea a la hora de realizar tests, ya que es pago el mandar mensajes, y en cuanto a mail, se implementó javaxMail.
- ➔ El hecho de mandar las notificaciones, queda a cargo del *Emisor de Notificaciones,* a quien todas las comunidades conocen y representa una de las funcionalidades del sistema.
- ➔ Hay 2 tipos de notificaciones, solicitud de revisión de incidentes, y reportes de incidentes (tanto de apertura como de cierre). Se especifica el tipo de notificación en el asunto del mensaje.
- ➔ En cuanto a los miembros de las comunidades, estos deben seleccionar su medio de comunicación (mail o whatsapp) de preferencia, y su forma de notificar (cuando suceden o sin apuros) de preferencia también. A partir de esta información, su email y su teléfono, se crea un receptor de comunicaciones, que se encarga de recibir las notificaciones destinadas a los miembros de las comunidades, y se las hace llegar a su medio seleccionado en el momento pertinente dependiendo de la forma seleccionada previamente. Estas selecciones pueden ser cambiadas posteriormente si el miembro así lo desea. Es esta clase *Receptor de Notificaciones,* quién se comunica con los adapters de las formas de los medios de comunicación. Es importante resaltar, que el miembro antes de llamar a su receptor para que reciba la notificación, este la filtra para ver si es de su interés, en caso contrario, no llama al receptor. Lo mismo ocurre con las solicitudes de revisión, se filtraran por distancia y por interés, pero siempre se enviaran cuando suceden. Se utilizó una técnica del tipo “pull”, ya que es el sistema quien está preguntando al usuario si está en rango para solucionar un incidente.

Servicios y Roles:

Los miembros de comunidad a la hora de indicar qué servicios son de su interés, deben informar su rol en cuanto a este, es decir, si se ven afectados por el mismo o si son meramente observadores de este. Para ello, se creó una clase que relaciona el rol y el servicio, simulando ser una tupla.
Es importante tener en cuenta, que esta situación puede cambiar en un futuro para un miembro en cuanto a un servicio, por lo tanto, son capaces de modificarlo.

Repositorio de Incidentes:

Se optó por contar con un repositorio de incidentes global, ya que es necesario conocer todos los incidentes, independientemente de a qué comunidad pertenece quien lo reportó, a la hora de generar los rankings (son independientes de las comunidades). Es importante tener en cuenta, que no se eliminan del repositorio aquellos reportes con antigüedad mayor a una semana, sino que se filtran a la hora de generar los rankings los reportes pertinentes.

Generación de Rankings:

Debido a que a la hora de generar los rankings, hay código/funcionalidades que se repiten entre ellos, con el fin de evitar esto y por el hecho que los ranking siguen una serie de pasos similares, se creó una clase abstracta llamada *Tierlist*, siguiendo un patrón de diseño template method para el método armarRanking. De esta manera, todos los rankings entienden el mismo método, que básicamente les dice que se creen, con sus diferencias en el funcionamiento interno. Esto ocurrirá al finalizar cada semana, y es el sistema, mediante el *Emisor de Notificaciones,* quien manda a generar los rankings.
Estos se crean en archivos CSV, y son posteriormente enviados a aquellas personas que las entidades prestadoras y organismos de control seleccionaron para recibir la información.

Cuarta Entrega - 15/09

Dominio:

Comunidades, Reporte de Incidentes, Incidente:

Se decidió modificar la estructura con respecto a las relaciones entre estas 3 clases. Se optó por qué las comunidades conozcan sus reportes (tanto de apertura como de cierre), y cada incidente conoce los reportes de apertura y cierre sobre el mismo. El primer reporte sobre una problemática (servicio de un establecimiento en particular), crea una instancia de un incidente. Luego, cada comunidad que reporte el mismo problema agregara otro reporte de apertura sobre la misma instancia de incidente. En el momento que una comunidad cierra el incidente, se considera cerrado en el sistema (importante a la hora de realizar los rankings), pero las comunidades que aún no lo cerraron lo consideran abierto para ellas, hasta que un miembro de su comunidad se los reporte como cerrado. De esta manera, una comunidad puede saber si abrió/cerró un incidente, si alguno de sus reportes está presente en la lista de reportes del incidente.

Lista de Reportes en Incidente:

Además, se decidió juntar los reportes de cierre y apertura de un incidente en una única lista de reportes, ya que de la otra manera generaba inconvenientes a la hora de persistir (tomaba cada reporte como de cierre y apertura a la vez).

Persistencia:

Repositorio de Empresas, Provincias y Municipios:

Estos 3 repositorios funcionan de una manera particular con respecto a los demás. Al iniciar el sistema, se deberá ejecutar un main en el que cada repositorio carga la base de datos con la información que le corresponde para luego permitir un correcto funcionamiento del sistema.

Repositorio Generico:

La gran mayoría de los repositorios, entienden los mismos métodos, y realizan la misma implementación. La diferencia se encuentra únicamente en la clase que recibe por parámetro dichos métodos a la hora de trabajar con la base de datos. Por ende, para evitar tanta repetición de código, se optó por crear un repositorio genérico que define los métodos y determina que dichos parámetros podrán ser de cualquier tipo de clase.

Servicios (Herencia):

Se decidió modelar la herencia en los servicios mediante el modelo de Single Table. Esto se hizo debido a que el comportamiento entre las subclases (Baño y Elevación)

era prácticamente el mismo. Por lo tanto, solo se consideró importante identificar de qué subclase se trataba a la hora de persistir. Esto a la vez, mejora la performance del sistema, al tener todos los servicios en una sola tabla.

Que persistir?

Se optó por persistir aquellas entidades que contienen atributos relevantes para el dominio, los cuales nos permiten lograr los requisitos de las funcionalidades propuestas. Por ello es que clases como medio de comunicación, forma de notificar, vía mail, entre otras, no se persistieron, al ser clases en su mayoría “Stateless”, es decir, utilizadas más que nada por su comportamiento.

API:

Coeficiente (CNF):

Dado que el servicio provisto por la API, debe realizar cálculos teniendo en cuenta un valor definido de manera arbitraria, se decidió incluirlo en el archivo CONFIG del sistema, y a la hora de realizar la consulta, se debe incluir. De esta manera, el servicio en la API, no quedará desactualizado en un futuro, si es que el valor del coeficiente llegase a cambiar, ni tampoco será difícil su modificación.