Diseño de Sistemas de Información – K3001/K3101

Trabajo Práctico Anual – Primera Entrega

Justificaciones a las decisiones de diseño

Grupo 9
22-4-2025

Tabla de contenido

Integrantes del grupo

- Lucas Pagnaro
- Juan Cruz Castelo
- Cynthia Abbate
- Paula González
- Bruno de Angelis

Ayudantes designados

- Gonzalo Turri
- Thomas Luca

Abstracciones en Clase Colección

Decidimos optar por una clase colección, la cual será posteriormente instanciada por administradores que utilicen nuestro sistema, con el fin de que estos sean quienes las crean en nuestro MetaMapa.

Dentro de esta clase, colocamos los atributos nombre y descripción del tipo String y una lista de hechos del tipo List, la cual posteriormente la instanciamos como un ArrayList que contendrá todos los hechos.

Hay dos principales abstracciones en esta clase, una lista de fuentes y el criterio de pertenencia.

Lista de Fuentes

Optamos por diseñarla de ese modo ya que, si bien en esta entrega nos enfocamos únicamente en las fuentes estáticas, consideramos que en un futuro, al implementar las fuentes dinámicas y de proxy, una colección podrá contar con más de una.

Cada fuente en particular implementa una interfaz Fuente, para tratar polimórficamente la importación de los hechos, sin importar de cuál provengan.

Criterio de Pertenencia

Cada colección puede tener su criterio de pertenencia, así diseñado para que, al momento de importar los hechos de una fuente, utilicemos el polimorfismo para decidir cada criterio.

Estos criterios de pertenencia están implementados como una Interfaz CriterioDePertenencia, la cual es utilizada por diversas clases, una por cada criterio considerado en la entrega, derivando la responsabilidad de identificar si un hecho pertenece o no a la misma.

Como una colección puede tener más de un criterio de pertenencia, en lugar de utilizar una lista, consideramos buena opción que una de las clases que utilicen esa interfaz sea CriterioCompuesto, la cual si tiene una lista de criterios y se encarga de identificar cuándo un hecho pertenece o no.

Al crear esta clase, y no tratarlo como una lista dentro de la colección, se consideró que le estaríamos designando la responsabilidad al criterio, abstrayéndose de la colección en sí. Al mismo tiempo, esta clase permite mayor flexibilidad al permitir que la colecciones tengan más de un criterio.

Filtrado de colecciones

Un visualizador puede filtrar hechos de una colección bajo una condición, como por ejemplo, solo visualizar los hechos de categorías Incendio. Esto se decidió emplear con

una interfaz Filtro, la cual es implementada por diversas clases según el filtro necesario, así como filtrado por fecha, filtrado por ubicación, por categoría y por título.

Decidimos el uso de esa interfaz para, nuevamente, un método en la clase Colección reciba un filtro y se encarga de definir polimórficamente cómo filtrar la lista de hechos, en lugar de hacer una cantidad inimaginable de IF, las cuales no solo serían poco flexibles, sino que dificultaría mucho la mantenibilidad.

Abstracción en los Hechos

Modelamos los hechos como una clase con varios atributos, estando entre ellos el título, la descripción, la fecha de acontecimiento y la fecha de carga. Además de estas hay varias más, las cuales serán tratadas a continuación como ítems para justificar cada decisión en particular.

Categoría y Etiquetas

Optamos por que categoría sea una lista de Strings, así al momento de compararlo en un filtro o criterio, se compara su contenido y listo. Inicialmente habíamos optado por que Categoría sea una clase con un atributo nombre (tipo String), pero su única función era esa, por lo que decidimos que con que sea únicamente del tipo String sería no solo suficiente, sino que más eficiente.

Las etiquetas tuvieron un proceso de diseño muy similar, una clase no nos era tan eficiente, así que optamos nuevamente por una lista de Strings.

Contenido Multimedia

Si bien en esta entrega no nos adentramos tanto en este aspecto, decidimos modelar el contenido multimedia como una clase, en la que tiene su path, una descripción y un enum con los posibles tipos de contenido. Aún no tiene comportamiento, pero decidimos ya diseñarlo para futuras entregas en las que sea necesario abstraer esta lógica.

Ubicación

La ubicación del hecho está contenida en una clase, con sus atributos longitud y latitud, abstrayendo así un poco la lógica de la clase Hecho, y logrando que, al compararlo con otra ubicación, esa cuenta se haga en esa clase y no en hecho, quien no debería tener esa responsabilidad.

Origen

Aunque en esta entrega solo utilicemos la carga manual, creamos un enum de origen para manejar sus tres posibles estados: manual, dataset y contribuyente.

Abstracción Solicitud de Eliminación

Se consideró que las solicitudes de eliminación sean una clase con el hecho a eliminar, el solicitante, el motivo, el estado de la solicitud (tipo enum con pendiente, confirmada y rechazada) y tanto las fechas de alta como de revisión de la misma.

¿Por qué son entidades propias?

Consideramos que sería una buena decisión modelarlas como una entidad propia en lugar de un simple método buscando lograr:

- Permisión de rastrear el historial de las mismas.
- Facilitar la implementación de flujos de aprobación como mensajes con pasos a realizar dentro de las mismas.
- Proporcionar una buena base para la auditoría y el cumplimiento normativo de nuestro sistema.

Bandera lógica estaEliminado en el hecho

Se implementó una flag en los Hechos, las cuales con un true o false nos permiten detectar si un hecho fue eliminado. Cuando un administrador confirme la eliminación del hecho, se seteará como true esta bandera.

Decidimos implementarlo de este modo para mantener un historial completo de los hechos que contienen las colecciones, permitir una recuperación de un elemento eliminado y facilitar autorías y análisis históricos.

¿Clase administrador?

En un momento nos habíamos planteado la utilización de una clase administrador para auditar quién fue el que confirmó/rechazó la solicitud, pero fue descartada ya que era una clase sin mucha utilidad y que no era muy necesario para el sistema según la consigna.

Conclusión logradas en el diseñado

Clase Usuario, heredada por Contribuyente, Visualizador y Administrador

En un principio consideramos la utilización de esa clase, la cual fue descartada porque no es tema del dominio representar el comportamiento del sistema, cómo y quien implementen cada acción. En futuras entregas se tendrá esto en cuenta y no utilizaríamos los “métodos botón”.

Descarte de Clase Categoría

Se descartó la implementación de una clase categoría ya que solo contenía un String con el nombre, y al momento de filtrar (para visualizar o para el criterio de pertenencia) se hacía muy complicado el compararlos, haciendo que reconsideremos su implementación y nos optemos por este tipo de dato String.