Justificaciones de Diseño - MetaMapa

Usuario

Durante el diseño del modelo, se analizaron dos enfoques principales para representar a los distintos tipos de usuarios del sistema: rol dinámico con permisos vs. herencia con clases concretas.

Opción 1 - Usuario con rol y permisos (modelo dinámico):

En este enfoque, se define una única clase **Usuario**, la cual posee un atributo **Rol**. Este rol contiene una lista de **Permisos** que determinan qué acciones puede realizar el usuario.

- **Ventajas:**
  - Gran flexibilidad y extensibilidad.
  - Permite agregar nuevos roles y modificar permisos dinámicamente, sin alterar la estructura del modelo.
- **Desventajas:**
  - Cada método requiere una verificación de permisos (if o validaciones).
  - Todos los usuarios comparten la misma interfaz (verían todos los métodos aunque no todos puedan usarlos).
  - Para tener diferentes comportamientos en un mismo método, se requerirían condicionales anidados.

Opción 2 - Usuario con herencia (modelo estático):

Se modela una clase abstracta **Usuario**, que representa lógica común a todos los actores, y se crean subclases concretas como **Contribuyente** y **Administrador** que encapsulan su propio comportamiento.

- **Ventajas:**
  - Cada tipo de usuario conoce solo sus propias responsabilidades (principio de interfaz mínima).
  - Permite definir métodos propios por tipo de usuario sin validaciones condicionales.
  - Más fácil de mantener en contextos donde los roles están claramente definidos y no cambian en tiempo de ejecución.
- **Desventaja:**
  - Menor flexibilidad: ante nuevos roles o cambios en los permisos se requiere modificar la estructura del código.

Decisión:

Finalmente optamos por el segundo enfoque (herencia) por las siguientes razones:

- El sistema actual contempla únicamente dos roles claramente diferenciados (**Contribuyente** y **Administrador**).
- No se prevé, en esta etapa, que los permisos cambien dinámicamente ni que se agreguen nuevos tipos de usuarios.
- Las responsabilidades de cada actor están bien delimitadas, lo que hace que las ventajas de la herencia (claridad, separación de responsabilidades, interfaz reducida) sean más relevantes que la flexibilidad del enfoque basado en roles.

Por estos motivos, el uso de una jerarquía de clases fue considerado más adecuado para cumplir con los requerimientos actuales del sistema, sin introducir complejidad innecesaria.
Visualizador.

El actor **visualizador** no fue modelado como una entidad o clase dentro del sistema, ya que **no posee estado propio ni comportamiento relevante** que deba representarse en el modelo de dominio. Solo accede a la plataforma y visualiza datos, lo cual es parte de la interfaz, no del dominio.

Si bien la consigna indica que cualquier persona puede aportar hechos, en el momento en que un visualizador decide hacerlo, **pasa a ser considerado un Contribuyente**. Por ese motivo, **ambos actores se unificaron en la clase Contribuyente**, que representa a toda persona que realiza un aporte, ya sea de forma anónima o identificada.

InfoContribuyente

Se decidió incluir una clase InfoContribuyente para encapsular los datos personales que una persona puede proporcionar al momento de aportar un hecho (nombre, apellido y edad). Esta información es opcional, y su existencia define si un hecho es anónimo o no.

Cada hecho almacena esta información como parte de su estructura, sin vincularse a un objeto Usuario o Contribuyente persistente.

Administrador

- Si bien aceptar y rechazar solicitud parecen botones, consideramos que implementan lógica de negocio y modifican el estado interno del sistema, ya que se debe cambiar el estado de visibilidad del Hecho y se deberá eliminar o persistir la solicitud.
- Más adelante, los hechos que se importen serán almacenados (persistidos) en una fuente de datos (perteneciente a otra capa del dominio).

Importador CSV

- Abstraemos la lógica necesaria para importar el contenido de los archivos de tipo .csv en una clase a fin de desacoplar y hacer a nuestro sistema más extensible, reusable y testeable.

Hecho de Texto

- Por trazabilidad agregamos una fecha de baja y un indicador de si está activo o no (ya que un hecho no se elimina del todo).
- Para esta primera entrega modelamos un único hecho, en el futuro haremos la distinción entre ambos tipos de hecho.
- En caso de que el hecho fuese aportado por un contribuyente, almacenamos su nombre, apellido y edad (en la clase InfoContribuyente), si dicho atributo es nulo es porque el contribuyente es anónimo o no fue aportado por contribuyente (origen distinto).
- Se aplicó el patrón Builder por su cantidad de campos obligatorios y opcionales (ubicación, categoría, fechas, autor).

Colección

- Si bien se nos dice que las colecciones están asociadas a una fuente de datos, consideramos que estas últimas serán bases de datos (o similar) por lo que no las modelamos en nuestro diagrama por pertenecer a otra capa del dominio.

Criterio de Pertenencia

- De esta interfaz heredarán los futuros criterios de pertenencia (clase) que desarrollaremos más adelante.

Categoría

- Suponiendo que en el futuro podrían crearse nuevas categorías, modelamos este atributo como una clase.

Ubicación

- Ubicación es un value object.

E1 -AFTER

Usuario, **Administrador** y **Contribuyente** no están más en nuestro diagrama debido a que no tienen comportamiento alguno por el momento.