UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

CÁTEDRA
DISEÑO DE SISTEMAS
DE INFORMACIÓN

MetaMapa: Sistema de gestión de mapeos colaborativos

Trabajo Práctico Anual Integrador
-2025-

ENTREGA 2: Arquitectura, y Modelado en Objetos - Parte II

Objetivos de la entrega

- Diseñar e implementar, de manera incremental, las nuevas funcionalidades.
- Incorporar nociones de ejecuciones de tareas asincrónicas y/o calendarizadas.
- Familiarizarse con otros componentes dentro de la arquitectura del Sistema y la orientación a servicios.
- Familiarizarse con diferentes estilos de integración con componentes y servicios externos.
- Familiarizarse con las APIs y su consumo.

Unidades del Programa Vinculadas

- Unidad 2: Herramientas de Concepción y Comunicación del Diseño
- Unidad 3: Diseño con Objetos
- Unidad 6: Diseño de Arquitectura
- Unidad 8: Validación del Diseño

Alcance

- Fuentes dinámicas.
- Fuentes proxy (intermediarias). Integración con sistemas externos
    - de otras ONGs,
    - de las propias instancias de MetaMapa.
- Rechazo de solicitudes de eliminación por tratarse de *spam* en forma automática.

Dominio

Fuentes dinámicas

En esta iteración, se implementará la funcionalidad que permitirá a los ***contribuyentes*** del Sistema subir ***hechos*** en forma anónima o registrada. Retomando el diagrama de arquitectura provisto en el Contexto general, este requerimiento deberá satisfacerse en el servicio de ***fuentes dinámicas***[1]. Si la persona no está registrada en la plataforma, podrá subir hechos sin posibilidad de edición posterior. En cambio, si los sube en forma registrada podrá realizar modificaciones al mismo en caso de que lo necesitara, pero solo en el plazo de una semana.

En ambos casos podrán estar sujetos a revisión por parte de las personas ***administradoras,*** que podrán aceptar, aceptar con sugerencia de cambios o rechazar la información. Se desarrollarán los mecanismos necesarios para gestionar la subida de información en formato tanto de texto como multimedia.

Fuentes proxy (intermediarias)

Existen sistemas externos provistos por otras organizaciones no gubernamentales que proveen hechos de generación propia mediante APIs y/o bibliotecas de diferente tipo. Aunque cada una de estas ***fuentes externas*** presentarán características propias y formatos diferentes, nos interesa diseñar un mecanismo

[1] En el diagrama de despliegue, se indica que el servicio de fuentes dinámicas tiene una base de datos de Hechos y Solicitudes. Como aún no tratamos bases de datos en la cursada, recomendamos implementar un Patrón Repositorio que cumpla esa función y mantenga las entidades necesarias en memoria.

general para integrarnos con ellas de forma unificada. Tal como se observa en el diagrama de arquitectura, esto se realizará a través de ***servicios*** y ***fuentes proxy*** (intermediarias).

Para poner a prueba este enfoque, deberemos dar soporte a dos nuevos componentes:

- **Fuente Demo:** una fuente que pueda dialogar con un sistema externo prototípico (y ficticio). En otras palabras, se trata de una integración con un sistema externo ficticio, para el cual contamos con una biblioteca cliente.
- **Fuente MetaMapa:** una fuente que nos permita dialogar con otras instancias de MetaMapa, mediante una interfaz REST.

Fuente Demo:

Dado que estas fuentes externas pueden tener interfaces muy diferentes entre sí —por ejemplo, bibliotecas complejas o APIs REST—, el objetivo es proponer un diseño flexible y extensible que pueda adaptarse a estos distintos modos de acceso y facilitar la obtención de hechos desde servicios externos.

Para cumplir con este objetivo, daremos soporte a un componente nuevo denominado ***fuente demo***, que se integrará con la siguiente interfaz (la cual será provista como una biblioteca externa):

```
 /** Hasta donde sabemos, no existe ninguna fuente externa que tenga esta interfaz. La misma es sólo a  que  modo de ejemplo y para poner nuestro diseño a prueba */
class Conexion {
   /**
    * Devuelve un mapa con los atributos de un hecho, indexados por nombre de
    * atributo. Si el método retorna null, significa que no hay nuevos hechos
    * por ahora. La fecha es opcional
   */
   Map<String, Object> siguienteHecho(URL url, DateTime fechaUltimaConsulta)
}
```

Es posible (y deseable, como se verá en futuras entregas) que más de una fuente retorne el mismo hecho, es decir, con iguales o muy similares atributos.

Debido a que este servicio externo es stateful, es decir, sus resultados dependen de los resultados previos, necesitamos que, una vez por hora, esta fuente proxy revise e incorpore nuevos ***hechos***, si hubiera. En otras palabras, el servicio externo no se consumirá en tiempo real.

Fuente MetaMapa:

Por otro lado, el Sistema deberá proveer la capacidad de comunicarse con otras instancias de MetaMapa. Esto lo haremos a través de una interfaz API REST, siguiendo la siguiente estructura tentativa (con posibles modificaciones en el futuro):

Endpoint	**Descripción**
GET /hechos	Esta ruta expone todos los hechos del sistema y los devuelve como una lista en formato JSON. La misma acepta parámetros para filtrar los

	resultados: categoria, fecha_reporte_desde, fecha_reporte_hasta, fecha_acontecimiento_desde, fecha_acontecimiento_hasta, ubicacion.
GET /colecciones/:identificador/hechos	Esta ruta, similar a la anterior, permite obtener los hechos asociados a una colección. Acepta los mismos parámetros y devuelve los resultados en el mismo formato.
POST /solicitudes	Permite crear solicitudes de eliminación, enviando los datos de la solicitud como un JSON a través del cuerpo *(body)* de la misma

Si bien en próximas iteraciones trabajaremos en exponer nuestros propios servicios utilizando esta interfaz, **por ahora se desea que nuestras fuentes proxy también sean capaces de consumir este tipo de API.** A diferencia del servicio anterior, este será consumido en tiempo real y se espera que sus resultados sean siempre actuales.

Colecciones

Para esta iteración, sigue válido el desarrollo de las colecciones realizado en la entrega anterior, pero se debe contemplar que a partir de esta entrega las colecciones podrán surgir también por fuentes dinámicas o fuentes proxy.

Además, para posibilitar la futura exposición REST de las mismas, se incorporará un atributo textual identificador ***(handle)*** que será un string alfanumérico, sin espacios, único entre todas las colecciones de un mismo servicio.

Rechazo de solicitudes de eliminación spam en forma automática

En la anterior entrega, se le permitió a las personas administradoras aceptar o rechazar las solicitudes de eliminación creadas por visualizadores y contribuyentes en forma manual. Para simplificar la tarea, se cuenta con un componente ya desarrollado que permite la detección de casos de spam evidentes. Si éste marca a la solicitud como *spam*, la solicitud deberá ser rechazada automáticamente.

```
interface DetectorDeSpam {
   boolean esSpam(String texto)
}
```

Requerimientos detallados

1. Como ***persona contribuyente***, deseo poder crear un hecho a partir de una fuente dinámica.
2. Como ***persona usuaria,*** quiero poder obtener todos los hechos de una fuente proxy demo configurada en una colección, con un nivel de antigüedad máximo de una hora.
3. Como persona usuaria, quiero poder obtener todos los hechos de las fuentes MetaMapa configuradas en cada colección, en tiempo real.

4. El Sistema debe permitir el rechazo de solicitudes de eliminación en forma automática cuando se detecta que se trata de spam.

Entregables

1. **Modelo del Dominio:** diagrama de clases que contemple las funcionalidades requeridas
2. **Justificaciones de Diseño**: Documento y Diagramas Complementarios.
3. **Implementación** de requerimientos de la Entrega Actual. Favorecer la reutilización del código de lo ya desarrollado.
4. **Diagrama de arquitectura** actualizado.
5. **Respuestas** a las siguientes preguntas, a modo de debate:
   - ¿Cómo se podría implementar un filtro de spam sin recurrir a servicios externos? Sugerencia: investigar sobre *el algoritmo TF-IDF para la detección de spam*.
   - ¿Cómo se podría implementar un filtro de spam en base a consumir servicios externos en la nube? ¿Qué consecuencias desde el punto de vista de costos y seguridad de datos traería?
6. **Análisis crítico de los requerimientos:** lean el artículo Designing for Accountability[2], analicen los requerimientos de la entrega anterior y la actual y debatan sobre las siguientes preguntas:
   a. ¿Tiene este sistema problemas desde el punto de vista de rendición de cuentas?
   b. ¿A quienes rinde cuenta este sistema? ¿A quienes no?
   c. ¿Hay partes interesadas que no hayan sido tenidas en cuenta en el diseño de este sistema?
   d. En base a las preguntas anteriores, ¿hay requerimientos que convendría modificar o incorporar?
7. **Bonus**: implementación de un filtro de spam básico que no requiera de un sistema externo.

[2] Acá encontrarás una versión traducida automáticamente