Apellido y Nombre: ..... Legajo: .....

Arquitectura (35)					Modelo de Dominio (35)				Persistencia (30)			Final
1a (15)	1b (5)	1c (5)	2a (5)	2b (5)	1 (15)	2 (5)	3 (5)	4 (10)	1 (15)	2 (10)	3 (5)	

- Se aprueba con **60** puntos y al menos un **50**% de cada punto
- Entregue cada punto (Arquitectura, Dominio y Persistencia) en hojas **separadas**, escriba en una sola carilla, ponga apellido y **numere cada hoja**
- **Lea todo el final antes de comenzar**

SIEEE

Contexto

Las plataformas SIEEE (Si [ocurre] **esto**, entonces [haz] **aquello**) o IFTTT (If This, Then That) permiten crear y programar acciones (llamadas recetas) para automatizar diferentes tareas y acciones en Internet. Su nombre deriva de su funcionalidad clave: cuando ocurre un evento específico (**esto**), la plataforma ejecuta una acción predeterminada en respuesta (**aquello**). SIEEE simplifica la integración de diferentes servicios y facilita la automatización de tareas diarias.

Dominio

En su núcleo, IFTTT opera a través de "recetas", que son combinaciones de una condición ("disparador") y una acción ("action"). La condición se establece en una aplicación o dispositivo y actúa como el desencadenante, mientras que la acción se lleva a cabo en otra aplicación o dispositivo como resultado de la condición cumplida. Los usuarios pueden crear y personalizar sus propias recetas o utilizar las creadas por otros miembros de la comunidad IFTTT.

Disparador / Conector

Para detectar un evento, deben configurarse “conectores”. Existen de 2 tipos, de consulta periódica (nosotros enviamos un pedido a un sistema externo cada cierto tiempo) y de “espera” (nuestro sistema da de alta un endpoint / url, a la cual un sistema externo debe enviarle un mensaje). En el primero, si se dispara un evento cuando se detecta que hay nueva información, pasan a evaluarse los filtros y a ejecutarse las acciones si corresponde; en el segundo, podemos asumir que si llega un mensaje, hay un nuevo evento.
Los conectores de consulta periódica, tienen: tipo, credenciales, periodicidad y detalle de la consulta (este último es un string que depende del tipo de conector). Los de ”espera”, tienen un endpoint (dominio + ruta), reciben mensajes http, con formato json.

Filtros / Condiciones

Los filtros reciben eventos de los conectores y se encargan de definir si se dispara la acción asociada a la receta. Éstos operan sobre los datos de los eventos, por ejemplo, si contienen un determinado texto, si el mensaje supera un tamaño “X” o si tiene un valor determinado en una determinada clave. Una receta puede no tener filtro. También se debe poder establecer condiciones lógicas sobre los

filtros, por ejemplo que se cumpla uno u otro, o que se cumplan 3 filtros, para poder disparar una acción.

Acciones

Una vez que se cumple la condición de un filtro debe ejecutarse una acción la cual consiste en enviar un mensaje de algún tipo. El mismo puede ser:

- Realizar una acción en alguna red social
- Enviar un mensaje por alguna mensajeria
- Realizar un pedido http a algún endpoint
- Dejar un mensaje en algún broker
- etc.

Es muy importante remarcar que para realizar la acción puede ser necesario algún dato del evento; por ejemplo, el emisor de un email, o un valor de una clave de un json.

Recetas

Las recetas se componen de un conector, cero o más filtros, y **una** acción a realizar. Cada persona usuaria del sistema puede tener una o más recetas, las cuales pueden estar activas o no.

Requerimientos

El Sistema debe:

1. Permitir el alta, modificación y baja de Recetas por parte de los usuarios.
2. Permitir que un usuario reutilice la receta creada y configurada por otro usuario.
3. Detectar la ocurrencia de eventos mediante los dos tipos de conectores: de consulta periódica y de espera.
4. Permitir el alta, modificación y baja de Filtros por parte de los usuarios.
5. Permitir que los Filtros contengan múltiples condiciones.
6. Permitir la configuración de una Acción sobre una Receta, por parte de los usuarios.

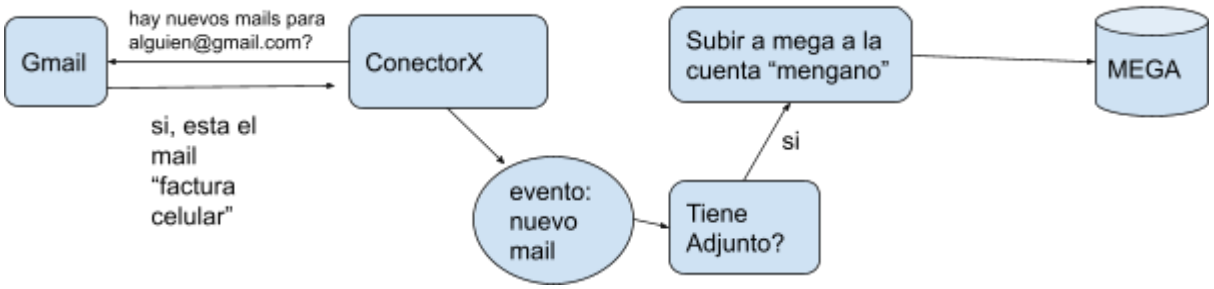
Otras consideraciones

- A futuro se deben poder agregar nuevos conectores fácilmente.
- La modificación y reutilización de recetas por parte del usuario debe ser lo más simple posible.
- Puede asumir que los eventos llegan y se procesan de a uno
- Las recetas tienen un único conector. Está fuera del alcance evaluar condiciones de 2 conectores distintos
- No hay ningún requerimiento que solicite datos sobre los eventos procesados por los usuarios, de hecho el acceso a los mismos post procesamiento del evento no debería permitirse **nunca**.

Ejemplos

	Bk de adjuntos	Apagado seguro	**Ofertas**
Conector	Se chequea la cuenta de mail X cada N tiempo para ver si hay nuevos correos	El sistema eléctrico nos envía un POST HTTP a http://miSIEEE.com.ar/receta/137 /evento cuando faltan 5 min para un	Se chequea la cuenta X de una red social cada N tiempo para ver si hay una nueva publicación

		servicio programado	
Filtro	Tiene adjuntos	Ninguno	contiene las palabras “venta” y “antiguo” en su descripción
Acción	Guardan los adjuntos en el storage X (MEGA, Dropbox, etc)	Se le envía un mensaje por SSH a las IPs [x,y,z] para que se apaguen.	Se envía un mensaje por telegram a la cuenta X, con el contenido de la publicación.



Esquema conceptual del ejemplo “Bk de adjuntos”

Arquitectura (35)

1. Proponga una arquitectura física para el sistema, mediante un diagrama de despliegue o esquema general:
    a. Deben figurar los componentes a utilizar, los sistemas externos, con qué protocolos todos se comunican, y cualquier aclaración que considere relevante. (15p)
    b. Tenga en cuenta que pueden llegar muchos eventos a la vez y que no queremos saturar nuestros recursos tratandolos todos al mismo tiempo. No apuntamos a manejar información crítica, si se demora “mucho” en realizar una acción o pierde algún paquete, los usuarios están cordialmente advertidos (5p)
    c. Se quiere que el usuario pueda comenzar a usar el sistema rápidamente, con lo cual se deben aceptar credenciales de redes sociales u otros sitios para registrarse/ingresar al sistema (5p)
2. Se observó que muchas de las recetas observan exactamente los mismos eventos, ya sea porque siguen las mismas cuantas en las redes sociales, u observan los mismos cambios en los recursos WEB, con lo cual los conectores tienen EXACTAMENTE los mismos datos. Teniendo en cuenta esto: (a) ¿qué se puede implementar para mejorar el rendimiento del sistema? Explique su funcionamiento (5p) (b) Dé un ejemplo concreto (5p)

Dominio (35)

Diseñar el modelo de dominio del sistema que resuelva los requerimientos aplicando el paradigma OO.

1. Armar la especificación del modelo utilizando diagramas UML (diagrama de clases obligatorio) (15p)
2. Justificar las decisiones de diseño que se tomen, por ejemplo, haciendo referencia a los principios que guían al diseño o las consecuencias de aplicar un determinado patrón. (5p)
3. Mostrar sobre un diagrama de objetos o esquema conceptual, el ejemplo de “**Ofertas**” del contexto (5p)
4. Explique con un diagrama de secuencia o pseudocódigo, cómo el escenario anterior procesa una publicación que tiene esas palabras “venta” y “antigüedad” en su contenido. (10p)

Persistencia (30)

Diseñar el modelo de datos del componente al sistema para poder persistir en una base de datos relacional a través de un ORM.

1. Armar la especificación usando un DER físico. Indicando las entidades, sus campos, claves primarias, las foráneas, cardinalidad, modalidad y las restricciones según corresponda. (15)
2. Justificar: (10)
    a. Qué elementos del modelo es necesario persistir y cuáles no
    b. Cómo resolvió los impedance mismatches.
    c. Las estructuras de datos que deban ser desnormalizadas, si corresponde.
3. El sistema se integrará contra uno de facturación a través de la base de datos. Se quiere que ello tenga un impacto mínimo sobre la operatoria del sistema (es decir no degrade su rendimiento para los usuarios). Dicho sistema de facturación quiere conocer cuantas acciones/eventos se procesaron por usuario y cuanto se demoro en cumplirlas, para cobrar por el servicio (5p)