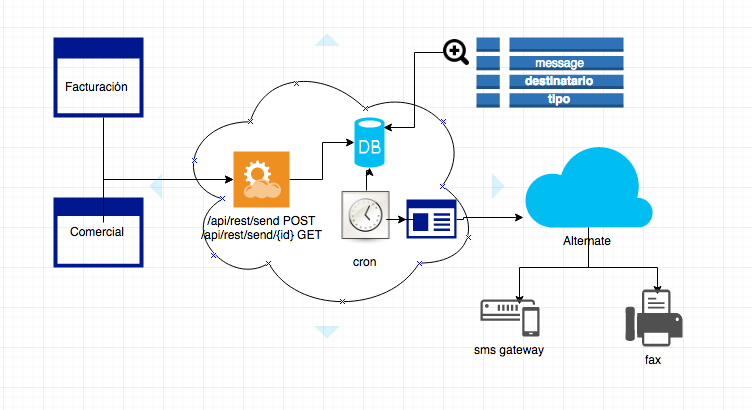
# Prueba técnica

## Introducción

Como prueba técnica, se solicita crear un servicio de envío de comunicaciones asíncronas para ser usado por cualquier aplicación de la compañía.

## Diagrama de arquitectura

La solución propuesta sería exponer un api que permita a los distintos clientes crear comunicaciones para ser enviadas por cualquier tipo de servicio y que se pueda consultar el estado de dichos envíos:



La idea es que las aplicaciones puedan construir los mensajes o comunicaciones que quieren enviar indicando destinatario (en formato número de teléfono en un principio, pero abierto a otro tipo como email), mensaje (limitado en tamaño o no dependiendo del tipo) y medio de envío (sms, fax inicialmente pero abierto a nuevos servicios o proveedores de otro tipo de medios de envío).

Una vez construido el mensaje este se almacenaría en una base de datos con un estado inicial y se contestaría a la aplicación con un id único. Un demonio en segundo plano estaría consultando la misma base de datos y se encargaría de ejecutar los envíos por el medio que se haya indicado reintentándolo un número configurable de veces en caso de error. Este demonio sería el encargado de contactar con el proveedor de medios de envío para realizar la petición efectiva de envío, esperar la respuesta y actualizar el estado del mensaje en la base de datos.

En cualquier momento, la aplicación que generó el mensaje puede consultar el estado del mensaje llamando al servicio con el id generado anteriormente.

## Definición de contrato de servicios

El api rest tendría las siguientes operaciones básicas

|  |  |  |
| --- | --- | --- |
| **Path** | **Tipo** | **Descripción** |
| /api/rest/send | POST | Servicio de creación y envío asíncrono de un mensaje.  Ejemplo:  {  "tipo": "sms",  "destino": “666778899”,  "mensaje": “La factura con 52,50€ de importe ha sido envíada a su banco”  }  Respuesta:  POST 200 Ok  {  “id”: “8393938949839”  } |
| /api/rest/send/:idComm | GET | Servicio de consulta de estado de un mensaje creado anteriormente  Ejemplo:  /api/rest/send/8393938949839  Respuesta:  {  "tipo": "sms",  "destino": “666778899”,  "mensaje": “La factura con 52,50€ de importe ha sido envíada a su banco”,  “estado”: [  “fecha”: “2017-05-09 12:02:02 GMT +2”,  “resultado”:”ko”,  “razón”: “el número no existe”,  “reintentos”: 2,  “reintentosPendientes”:5  } |

## Propuesta de mejoras

Correcciones

Con respecto al contrato propuesto de ejemplo para enviar mensajes, veo varios problemas

1. Para seguir la metodología RESTFUL no se debe hacer una llamada GET para realizar una modificación. Si somos estrictos, envíar un mensaje es realizar una modificación – escribir un mensaje en la base de datos y enviar un mensaje - aunque hay que reconocer que cumple el requisito de “idempotencia” (si es que esta palabra existe): cada vez que se ejecute obtendremos el mismo resultado.
2. Seguridad: se va a enviar el número de teléfono en claro en la url así como el mensaje. Deberíamos enviarlo por post y cifrarlo.
3. Relacionado con lo anterior, las urls que se pueden construir tienen un tamaño máximo por lo que no es buena idea enviar el mensaje en la url.
4. Pensando en el envío por fax, tampoco podríamos enviar imágenes a no ser que las transformemos a base64 .

Mejoras

* **Control de acceso al servicio**: para controlar quién puede hacer uso del servicio habría que habilitar una lista de blanca de aplicaciones autorizadas a usarlo. Se podría implementar con un token en la request.
* **Control de peticiones por segundo**: controlar el número de peticiones por aplicación y por segundo que puede servir para evitar ataques DDOS.
* **BBDD distribuida**: con este planteamiento, la base de datos es un cuello de botella limitando el número de mensajes que se podrían crear simultáneamente a pesar de cuántas instancias del servicio se desplieguen. Con una base de datos distribuida en cada instancia con algún mecanismo de replica entre ellas mejoraríamos en este aspecto.