DOCUMENTO DE RIESGOS

Arquitectos cognitivos:
- Jesús Sebastián Tocas Atarama
- Pablo Atahonero García de Blas

Nosotros, como arquitectos cognitivos, tuvimos una reunión con los Arquitectos Senior para debatir y cuestionar las siguientes decisiones de diseño que se han tomado.

4º. Decisión de Diseño ADD-04: Identificación del usuario

Esta decisión arquitectónica está relacionada con el requisito funcional:
RF003-Identificación de usuarios.

En relación a esta decisión de diseño habíamos considerado utilizar una base de datos relacional para almacenar la información del usuario, tanto su correo como su contraseña. Por motivos de seguridad para evitar los ataques de fuerza bruta habíamos pensado en incluir un número máximo de intentos de inicio de sesión por parte de un usuario en intervalos de 1 hora.

Por parte de los ASC, nos parece buena idea el uso de la OTP (One Time Password) para el caso de los usuarios que no accedan al sistema de forma habitual. Pero para aquellos usuarios que tengan que ingresar con más frecuencia, puede ser una molestia en cuanto a complejidad y tiempo para acceder al sistema, ya que requeriría algún paso más, y un cierto tiempo hasta que el usuario pueda recibir la contraseña temporal.
En el caso de que tengamos que obtener esta contraseña vía sms, puede vulnerar la seguridad de nuestra cuenta al ser sensible a ataques como SIM Swapping.

En cuanto al aspecto de implementar la verificación en dos pasos, vemos que es un tanto redundante, ya que el concepto es muy parecido al de la aplicación de la OTP en nuestro sistema, por lo que hemos considerado ignorar dicha decisión.

5º Decisión de Diseño ADD-05: Comunicación de las BBDD de los microservicios mediante el patrón CQRS

Esta decisión arquitectónica está relacionada con el requisito funcional:
RF005-Acceder a base de datos.

En lo referente a esta decisión de diseño no podemos argumentar nada ya que es un requisito que se pide explícitamente en el enunciado, por lo que estamos de acuerdo con esta decisión.Sin embargo, hemos encontrado algún inconveniente que puede ser un problema a la hora implementar esta tarea, ya que es un patrón que no hemos podido estudiar en profundidad, además nos hace retrasar el desarrollo del proyecto.

6º Decisión de Diseño ADD-06: Usar BBDD para mostrar un catálogo de microservicios

Esta decisión arquitectónica está relacionada con el requisito funcional:
RF004-Catálogo de microservicios.

Respecto a esta decisión, estamos de acuerdo con los Arquitectos Senior aunque también habíamos considerado el patrón Composite ya que se pueden introducir nuevos tipos de elementos sin descomponer el código que ya teníamos ,que ahora ya funciona con un árbol.

7º Decisión de Diseño ADD-007:Uso del patrón Composite para mostrar los microservicios

Esta decisión arquitectónica está relacionada con el requisito funcional:
RF004-Catálogo de microservicios

Como inconveniente puede resultar difícil proporcionar una interfaz común para microservicios distintos. En algunos casos,habría que generalizar en exceso la interfaz común dicha anteriormente, provocando que sea más difícil de comprender.

Aunque no habría que descartar por completo ya que tiene aspectos muy buenos a tener en cuenta, como que:

- Hace más fácil añadir nuevos tipos de microservicios.
- Simplifica las cosas al cliente, porque ellos pueden tratar los objetos simples y compuestos de forma uniforme.

Pese a los puntos positivos anteriormente redactados hemos decidido junto con los Arquitectos Senior rechazar esta decisión para mantener la decisión ADD-006,que sería la más adecuada para cubrir el RF-004-Catálogo de microservicios.

8º Decisión de Diseño ADD-007:Uso de un gateway a diferentes APIs

Esta decisión arquitectónica está relacionada con los requisitos funcionales:
RF001.1-Procesamiento de pedidos por parte de la app
RF001.2-Consulta de actualizaciones.
RF001.3-Comunicación de clientes con microservicios

En lo referente a esta decisión de diseño tampoco podemos argumentar nada ya que es un requisito que se pide explícitamente en la práctica, por lo que estamos de acuerdo con esta decisión.
Aun así, hemos localizado una serie de inconvenientes que se podrían presentar a la hora de implementar esta decisión en nuestra arquitectura software:

-El proceso de programación de un gateway sigue un proceso bastante rígido y por lo tanto no se programa con facilidad.
-La solución de problemas del gateway puede ser difícil, ya que se necesitan herramientas diferentes para equipos con distintos protocolos.
-En el momento que un gateway falla, se pierde la comunicación con la red. Esta comunicación no se puede restablecer hasta que se halle el problema, por lo tanto, va a

través de cada equipo de la red y conlleva tiempo la resolución de problemas de forma individual hasta que se encuentra dicho problema.

9º Decisión de Diseño ADD-009:Canal de mensajería mediante un event bus

Esta decisión arquitectónica está relacionada con los requisitos funcionales:
RF002-Canal de mensajería
RF002.2-Propagación de actualizaciones entre microservicios

Con respecto a esta decisión de diseño, no tenemos nada que objetar ya que este requisito está bien reflejado en la práctica, por lo que estamos de acuerdo con esta decisión.
Sin embargo, cuando tengamos que trabajar con esta arquitectura, nos enfrentaremos a unos cuantos problemas que es muy conveniente tenerlos en cuenta:

-Tienen un alto consumo de memoria, ya que cada microservicio tiene sus recursos y sus bases de datos,y por lo tanto necesitan más memoria y procesamiento.

-Se necesita mucho tiempo al principio. Cuando empezamos a crear la arquitectura basada en eventos, necesitamos más tiempo y desarrollo para separar los microservicios, implementarlos y además comunicarlos entre sí.

-Complejidad en la gestión.Si controlamos un número elevado de microservicios, será más difícil dirigir la gestión e integración de los mismos. Hacen falta herramientas avanzadas para poder manejar al mismo tiempo diferentes microservicios.

10º Decisión de Diseño ADD-010:Hacer que el gateway procese los pedidos

Esta decisión arquitectónica está relacionada con los requisitos funcionales:
RF001.1-Procesamiento de pedidos por parte de la app.

Anteriormente hemos localizado una serie de inconvenientes que se podrían presentar a la hora de implementar esta decisión en nuestra arquitectura software mediante un gateway en la Decisión de diseño *ADD-008:Uso de un gateway a diferentes APIs:*

-El proceso de programación de un gateway sigue un proceso bastante rígido y por lo tanto no se programa con facilidad.
-La solución de problemas del gateway puede ser difícil, ya que se necesitan herramientas diferentes para equipos con distintos protocolos.
-En el momento que un gateway falla, se pierde la comunicación con la red. Esta comunicación no se puede restablecer hasta que se halle el problema, por lo tanto, va a través de cada equipo de la red y conlleva tiempo la resolución de problemas de forma individual hasta que se encuentra dicho problema.

En este caso al proponer de nuevo un gateway tendríamos los mismos problemas e inconvenientes que en la decisión *ADD-008:Uso de un gateway a diferentes APIs.* Además el procesamiento de pedidos mediante el uso del gateway es muy lento y por ello hemos decidido conjuntamente rechazar esta decisión de diseño.

11º Decisión de Diseño ADD-011:Implementar el procesado de pedidos como un microservicio

Esta decisión arquitectónica también está relacionada con los requisitos funcionales: RF001.1-Procesamiento de pedidos por parte de la app.

Tras haber leído esta decisión, podemos sacar unas cuantas conclusiones y aspectos del uso de un microservicio para el procesado de pedidos, unas positivas y otras negativas al respecto:
Aspectos positivos:

- Gran agilidad
- Mejor escalabilidad
- Más fácil de implementar

Aspectos negativos:

- Rendimiento más pobre, ya que los microservicios necesitan comunicarse.
- Más difícil de mantener la red.
- Problemas de seguridad.

Tras hacer una valoración de estos factores, podemos decir que estamos de acuerdo con esta decisión de diseño.

12º Decisión de Diseño ADD-012:Elección de contenedor de nuestros microservicios, con Docker

Esta decisión arquitectónica también está relacionada con los requisitos funcionales: RF002.1.1-Uso de contenedores de microservicios para orquestar contenedores.

En correspondencia con esta decisión de diseño ADD-012, hemos realizado una investigación de la herramienta Docker para determinar si es adecuada para dicha decisión. Hemos hallado algunos obstáculos que pueden ser un inconveniente para implementarlo:

- El almacenamiento de datos persistentes es complicado. Debido a su diseño, todos los datos dentro de un contenedor desaparecen si se apaga el contenedor, a menos que lo almacenes en otra parte.

- No todas las aplicaciones se benefician de los contenedores. Sólo las aplicaciones diseñadas para funcionar como un conjunto de microservicios discretos son las que más se benefician de los contenedores por lo general.
- Los contenedores no funcionan a velocidades normales. Los contenedores consumen recursos de manera más eficiente que las máquinas virtuales. Pero los contenedores todavía están sujetos a una sobrecarga de rendimiento debido a la superposición de redes, la interfaz entre los contenedores, etc.

Finalmente, tras reflejar los impedimentos que conlleva usar Docker, consideramos que es una buena opción su uso dado que es una de las herramientas pioneras para la organización de los contenedores.