Decisiones

Arquitectura de software

Status:

[proposed]
[ADR-0001]

Decisores:

Los integrantes del grupo 4

Date:

[2022-11-02]

Context and Problem Statement

Se nos propone el diseño de un sistema con muchos datos entrantes, ¿cómo podemos hacer que esta cantidad de datos se comunique con el resto de nuestro sistema y cumpla sus necesidades?

Decision Drivers

RF1, RF1.1, RF2, RF3, RF3.1, RF4, RF5, RF5.1

Considered Options

[Implementación de una Arquitectura de Microservicios]
[Implementación de una arquitectura basada en eventos]

Decision Outcome

Opción elegida: "[Implementación de una arquitectura basada en eventos]", porque permite que varios subsistemas procesen los mismos eventos proporcionadas por los dispositivos IoT, y procesados en tiempo real.

Positive Consequences

-Procesado del flujo de datos entrante en tiempo real. Los dispositivos IoT que incluye el Sistema nos van a proporcionar un flujo de datos grande, y esta arquitectura es perfecta para poder procesar todos estos datos adecuadamente. Además, permite controlar todos estos datos en tiempo real con analíticas instantáneas.

-Escalabilidad. Comparado con arquitecturas monolíticas, una arquitectura distribuida como esta nos permite escalar dependiendo de las necesidades de los datos, independientemente de la cantidad de datos o la cantidad de eventos que deban ser procesados. Al ser una arquitectura desacoplada, se puede ver qué módulo o servicio necesita ser escalado porque esté escaso de recursos, sin tener que escalar todo el sistema, por ello la escalabilidad se consigue de forma natural.

-Independencia de servicios. Los diferentes servicios no necesitan si quiera conocerse o depender unos de otros, los eventos operan independientemente, por lo que reducimos la responsabilidad de cada uno a que hagan solo su trabajo. Esto beneficia en tiempo a la hora de tener que probar o actualizar un servicio, ya que solo hay que hacerlo en dicho servicio.

-Resistencia a fallos. Los eventos son asíncronos, luego si un servicio cae, el resto del sistema seguirá funcionando, y el servicio que tuvo el fallo, podrá ejecutar la cola de eventos que tenía, sin perder información en el proceso.

Negative Consequences

-Encontrar los fallos. Un software de este tipo tiene muchos productores de eventos, así como consumidores, por lo que encontrar un fallo en la arquitectura puede suponer un reto, al no saber toda la posible información que nos puede llegar por todos los dispositivos IoT. Es muy difícil anticipar el comportamiento de entradas que todavía ni conocemos.

-Complejidad. Como todas las arquitecturas basadas en eventos, son arquitecturas muy complejas de implementar, así como de encontrar errores, probar y desplegar.

Otras opciones

[Implementación de una Arquitectura de Microservicios]
* Buena, por la posibilidad de escalar rápidamente e independencia de servicios.
* Mala, por no ofrecer la baja latencia y balanceo de carga que nos ofrece la arquitectura basada en eventos.

* Mala, por no ser rápida procesando grandes cantidades de datos.
* Mala, porque hay más posibilidad de fallar en la comunicación con otros servicios.

Elección de algoritmos

Status:

[proposed]
[ADR-0002]

Decisores:

Santiago Arias - ASS

Date:

[2022-11-04]

Context and Problem Statement

Necesitamos elegir entre dos algoritmos, uno para optimizar el volumen de órdenes de trabajo, y otro para predecir el fallo en una línea de trabajo y asignar recursos a otras.

Decision Drivers

RFN – El sistema debe seleccionar el algoritmo a utilizar de forma autónoma.

Considered Options

[Patrón de diseño Strategy]

Decision Outcome

Opción elegida: "[Patrón de diseño Strategy]", este patrón de diseño nos permite cambiar cómo se comporta un módulo en tiempo de ejecución, en este caso se refiere al módulo que

se encarga de asignar el trabajo a las distintas líneas de trabajo, que gracias a una clase abstracta que contendrá como hijos las subclases con cada algoritmo, podrá ir intercalándolos según lo necesite, utilizando una implementación u otra.

Positive Consequences

-Espacio para mejorar. Al ser los algoritmos clases independientes, si encontrásemos un algoritmo que funciona mejor, podríamos simplemente implementarlo y funcionaría sin tener que alterar todo el sistema.

-Poder alternar los algoritmos en tiempo de ejecución. La razón principal por la que necesitamos este patrón. El código contiene las instrucciones necesarias para elegir entre los dos algoritmos en tiempo de ejecución.

Negative Consequences

-Añade dos objetos y una interfaz al software, para implementar algo relativamente sencillo.[1]

Patrón Observer para las notificaciones y suscripciones

Status:

[proposed]
[ADR-0003]

Decisores:

Santiago Arias - ASS

Date:

[2022-11-04]

Context and Problem Statement

Necesitamos notificar a los operarios de la factoría, así como permitirles suscribirse a diferentes eventos y notificaciones.

Decision Drivers

RF3, RF3.1

Considered Options

[Patrón de diseño Observer]

Decision Outcome

Opción elegida: "[Patrón de diseño Observer]", este patrón de diseño nos permite notificar a los objetos suscriptores (los operarios), de actualizaciones en los objetos a los que se han suscrito (eventos).

Positive Consequences

-Fácil de implementar. El objeto al que se pueden suscribir contendrá una lista de los operarios que se han suscrito. Los operarios, a su vez, tienen en su clase un método que podemos llamar *notificar*, que será al que el objeto suscrito llamará cuando se produzca alguna actualización que requiera ser notificada.

-Bajo coste computacional. Las llamadas se realizan automáticamente, no hace falta que haya un objeto esperando un cambio, sino que el propio objeto que cambia se encarga de notificarlo.

-Bajo acoplamiento. Sigue el principio de bajo acoplamiento entre clases que interactúan entre ellas.

-Se pueden añadir y quitar observadores en cualquier momento.

Negative Consequences

-Fugas de memoria. Si al cancelar una suscripción esta falla, este observer se convierte en una referencia que le impide ser eliminada por el colector de basura en lenguajes como Java, o incluso un error completo en lenguajes sin colector de basura. Este error es mitigable implementando una buena función de cancelación de suscripción. [2]

Mostrar las analíticas

Status:

[proposed]
[ADR-0004]

Decisores:

Ángel Covarrubias - ASC

Date:

[2022-11-05]

Context and Problem Statement

Necesitamos poder mostrar las analíticas en tiempo real del proceso productivo y las órdenes de trabajo.

Decision Drivers

RF1, RF1.1

Considered Options

[Patrón de diseño Modelo Vista Controlador]

Decision Outcome

Opción elegida: "[Patrón de diseño Modelo Vista Controlador]", con este patrón aseguramos que la lógica está separada de su visualización.

Positive Consequences

-Bajo coste computacional. Si no se aplica un MVC en este caso, no tendría sentido la cantidad de poder computacional que necesitaríamos para poder abrir una simple interfaz, teniendo un controlador dedicado específicamente para sacar los datos y procesarlos por

pantalla, reducimos esta necesidad computacional, así como tiempos de carga y mejoraríamos la visión de estadísticas en tiempo real.

-Separación de componentes. Se asegura de que la lógica va a estar separada de la visualización de analíticas ahorrándonos futuros problemas.

Negative Consequences

-No se podría implementar si los datos recibidos del proceso productivo no llegan correctamente, al igual que si las ordenes de trabajo no son las adecuadas producirían que los datos expuestos en las analíticas sean incorrectos.

Asignaciones de órdenes de trabajo

Status:

[proposed]
[ADR-0005]

Decisores:

Ángel Covarrubias - ASC

Date:

[2022-11-06]

Context and Problem Statement

Necesitamos poder atribuir el orden de trabajo de los operativos y de las máquinas.

Decision Drivers

RF2

Considered Options

[Patrón de diseño Mediator]

Decision Outcome

Opción elegida: "[Patrón de diseño Mediator]", con este patrón aseguramos que usuario y los componentes de trabajo no se comunican de forma directa.

Positive Consequences

-Comunicación correcta. Al haber un mediador entre los dos no hay problemas de comunicación ya que no es necesario de que ambos estén disponibles para que la orden sea comunicada. El usuario decide el orden de trabajo, el mediador guarda esa información y cuando el operativo o la maquina este disponible se le mandara el mensaje.

Negative Consequences

-No se podría implementar si se quiere comunicar directamente el sistema principal con los operarios o las maquinas.

Centro de notificaiones

Status:

[proposed]
[ADR-0006]

Decisores:

Ángel Covarrubias - ASC

Date:

[2022-11-05]

Context and Problem Statement

Necesitamos recibir los datos de los sensores, visualizar las analíticas y operar todas las funcionalidades del sistema.

Decision Drivers

RF1, RF1.1

Considered Options

[Patrón de diseño Mediator]

Decision Outcome

Opción elegida: "[Patrón de diseño Mediator]", este patrón va a ayudar a recibir los datos generados por los sensores y visualizar las analíticas creadas a partir de esos datos, separando estas dos funcionalidades.

Positive Consequences

-Comunicación correcta. Utilizando este patrón de diseño conseguiremos que haya un mediador entre los datos y la visualización de analíticas. De esta forma hay un mayor control en el sistema en cuanto a la gestión de la información y la creación de toda la estadística.

Negative Consequences

-No se podría implementar si alguna de las dos partes falla, es decir, si no se reciben datos de los sensores o si la visualización de las analíticas da algún error.

Comunicación entre dispositivos IoT:

[proposed]
[ADR-0009]

Decisores:

Vicente González - ASC

Date:

[2022-11-06]

Context and Problem Statement

Existe una familia de dispositivos IoT compuesta por tres sensores en los que el primero envía información al segundo y este al tercero que finalmente lo envía al centro de notificaciones.

Decision Drivers

RFN – El sistema debe seleccionar el algoritmo a utilizar de forma autónoma.

Considered Options

[Patrón de diseño Chain of Responsability]

Decision Outcome

Opción elegida: "[Patrón de diseño Chain of Responsability]", es un patrón de diseño de comportamiento que te permite pasar solicitudes a lo largo de una cadena de manejadores. Al recibir una solicitud, cada manejador decide si la procesa o si la pasa al siguiente manejador de la cadena.

Positive Consequences

-Controlar el orden de los pasos de manera dinámica.

- Desacopla en clases la invocación de las operaciones y la realización de la propia acción. Pudiendo testear fácilmente cada acción por separado.

Negative Consequences

Un error en el reenvío puede resultar en mensajes perdidos.

Monitorizar el estado de cada familia de dispositivos IoT: //A medias

[proposed]
[ADR-0010]

Decisores:

Vicente González - ASC

Date:

[2022-11-06]

Context and Problem Statement

Existe una familia de dispositivos IoT compuesta por tres sensores en los que el primero envía información al segundo y este al tercero que finalmente lo envía al centro de notificaciones.

Decision Drivers

RF1, RF1.1

Considered Options

[Patrón de diseño Adapter]

Decision Outcome

Opción elegida: "[Patrón de diseño Adapter]", es un patrón de diseño de comportamiento que te permite pasar solicitudes a lo largo de una cadena de manejadores. Al recibir una solicitud, cada manejador decide si la procesa o si la pasa al siguiente manejador de la cadena.

Positive Consequences

-Controlar el orden de los pasos de manera dinámica.

- Desacopla en clases la invocación de las operaciones y la realización de la propia acción. Pudiendo testear fácilmente cada acción por separado.

Negative Consequences

Un error en el reenvío puede resultar en mensajes perdidos.

Base de datos

[proposed]
[ADR-0011]

Decisores:

Irene Pérez Santiago - ASS

Date:

[2022-11-07]

Context and Problem Statement

La BBDD deberá almacenar además de las órdenes de trabajo el inventario del material existente

Decision Drivers

RF1, RF1.1

Considered Options

Patrón de Base de Datos por Micro-Servicio

Decision Outcome

Opción elegida: "[Patrón de Base de Datos por Micro-Servicio]", es un patrón que al elegir la Arquitectura de Micro-Servicios nos permite un acoplamiento flexible a la estructura ya que esta manera cada microservicio, en este caso tendríamos ordenes de trabajo e inventario, cada una de ellas se puede almacenar de manera independiente sin afectar al otro.

Positive Consequences

-Independencia

-Permite que se puede almacenar y recuperar de manera distintas

-No se comparte la capa de datos, así que un microservicio no puede acceder a otro.

Negative Consequences

En caso de que sea necesario que se transmitan información de un microservicio a otro no sería posible

Monitorizar el estado de cada familia de dispositivos IoT: //A medias

[proposed]
[ADR-0012]

Decisores:

Irene Pérez Santiago - ASS

Date:

[2022-11-07]

Context and Problem Statement

Es deseable el uso de medidas de seguridad por determinar para el acceso de los usuarios.

Decision Drivers

RF1, RF1.1

Considered Options

Patrón Access Token

Decision Outcome

Opción elegida: "[Patrón Access Token]", es un patrón que permite securizar los inicios de sesión, un token de acceso es un string aleatorio que identifica a un usuario y puede ser utilizado por la aplicación para realizar llamadas API. El token incluye información sobre cuándo expirará el token y qué aplicación generó el token.

Positive Consequences

-Permite establecer un método de seguridad

Negative Consequences

Puede fallar y que la seguridad quede comprometida

Monitorizar el estado de cada familia de dispositivos IoT: //A medias

[proposed]
[ADR-0013]

Decisores:

Irene Pérez Santiago - ASS

Date:

[2022-11-07]

Context and Problem Statement

Los sensores IoT se clasifican en tres familias, cada una de las cuales comparte cierta funcionalidad, pero dispone de otras diferentes entre una familia y otra.

Decision Drivers

RF1, RF1.1

Considered Options

Decision Outcome

Positive Consequences

Negative Consequences