APACHE LOG 4J

Una buena práctica es poner logs. Un log es un mensaje de depuración nuestra que toman importancia. Por ejemplo una consola de un terminal dentro de una página web en un entorno de ejecución donde aparecen en la consola todo tipo de mensajes, mensajes que el usuario no ve pero que el programador de esta página accede a ellos para ver cómo se ejecuta y que pasa en la página.

System.out.printl no debe aparecer por pantalla ya que puede ser un problema. El usuario no debe verlo, pero los mensajes deben de seguir apareciendo ya que necesitamos saber que pasa en cada momento.

¿Entonces la información donde va a aparecer? LOG4J es una librería que establecen un paintline en el cual cuando mandas un mensaje de depuración la librería lo guarda donde tú se lo has pedido y lo hace de un formato determinado añadiendo unos datos determinados.

En el diagrama UML hay una clase principal que usamos para mandar mensajes al log, además hay unas clases de configuración de cómo se guardan, etc. A nivel de código funciona añadiendo en java la librería de LOG4J, luego en las clases que nosotros hagamos vamos a tener una línea donde decimos que queremos un loger, que lo llamamos como queramos y decimos que ese loger va a ser el que me dé el logmanager(una de las clases), a partir de ahí lo usas como quieras. Usando loger tengo una serie de métodos que puedes meter como LOGGER.INF, LOGGER.WORN, LOGGER.ERROR, etc. Eso lo voy a utilizar para que cuando los mensajes aparecen vayan clasificados. Además, puedes hacer que el programa solo te de los errores para que sepas que debes corregir.

Los logs nos ofrecen una manera de gestionar la información, donde se puede establecer que cada clase guarde sus mensajes en un fichero distinto, los logs pueden tener un tamaño máximo y cuando llegues, se vaya eliminando las primeras líneas o generar uno fijo para que cuando se llene cree otro para seguir organizándolo.

Cuando entras en un servidor web, debes poder gestionar quien entra y los cambios que puede realizar.

Los logs no se borran nunca por tema técnico, legal y por dinero. Puedes analizar los patrones de uso de los elementos para los distintos usuarios.

Hay iniciativas nuevas que es usar los logs de forma organizada para tener telemetría digital de las aplicaciones. Ejemplo: OpenTelemetry. Nosotros la usamos para ver cómo se están usando nuestros programas. La primera telemetría se usaba para medir el tiempo de respuesta de una página web. Lo ideal es como mucho 1 décima de segundo. (cuanto tarda en ejecutarse cada componente interno). Es importante ya que cada vez que se ejecuta cada método de un programa puedes extraer cada vez que se llama al método, si todo está bien, la estructura, etc.

BIG DATA

El termino es hijo de DATA WAREHOUSE que a su vez es hijo de la ANALÍTICA que además es hijo de B.P OPERACIONALES.

La base de datos operacional puede soportar analíticas, pero llega un momento que como trabaja con tantos datos pues se satura la base de datos operacional y se genera el problema de que no hay capacidad de ejecución suficiente. Entonces data warehouse es la traslación de

los datos mediante sistemas transaccionales y mediante OLAP a una base de datos especifica solo para hacer analíticas.

Gracias a esto que funciono tan bien que ganaron mucho dinero, pero en la época de 2000 a 2010 hubo varios navegadores (GOOGLE, YAHOO, AMAZON) que tenían un problema, sus modelos de negocio implicaban millones de clientes generando muchos datos que no podían ir a la base de datos generacionales porque eran logs, ahí aparece el big data que es coger los navegadores y unirlo al resto de la cadena. El camino del recorrido de la cadena se llama datos estructurados y el recorrido del navegador al big data conectada a la cadena se llaman datos semiestructurados o no estructurados. El problema es que después además de logs puedes tener post o fotos, entonces el big data coge todo y lo añade al piteline de procesamiento que nos da lugar a los DATALAKE que es la última evolución antes de DATA LAKEHOUSE.

FRAMEWORDS Y COMPONENTES

Los informáticos tratamos de facilitarle la vida al usuario. La herramienta que usamos para reducir el espacio entre el usuario y el técnico son las interfaces de usuario.

Las nuevas profesiones que se crean destruyen profesiones como administrativas, etc. Lo que hacemos tienen consecuencias muy palpables y que nos pueden perjudicar a nosotros y las herramientas que nosotros creamos para el bien, su objetivo último es la automatización de tareas y, por tanto, hay tareas que entonces ya no vas a tener que realizar.

Vamos a un mundo en el que el nivel de automatización va a dejar a muchísimas personas que se dedican a determinadas profesiones sin trabajo.

El capitalismo se basa en la ineficiencia para poder crear beneficio porque al añadir valor entonces reduces más la ineficiencia que tu enemigo. Un sistema informático rinde siempre igual, no se cansa y contra eso es imposible vencerle.

Siempre estamos en un negocio de veneficios decrecientes, siempre vas hacia abajo, las máquinas no. Lo que va a pasar es que habría muchas inteligencias artificiales con un sistema que se adaptará a nosotros.

La visión de futuro es que todas las mejoras que hemos ido viendo en campos técnicos y que han dado una reducción de la ineficiencia y una reducción de trabajadores, con inteligencia artificial eso se multiplica y va a afectar a las profesiones de cuello blanco y cuello azul es decir profesiones que no se han podido hasta hoy automatizar, médicos, abogados, nosotros…

Las interfaces de usuario evolucionan muy rápido. Los usuarios se pueden llegar a cansar de programas y al renovarse vuelven a aceptarlo. Ejemplo: los coches tienen un tiempo de comercialización de 10 años porque la tecnología en ese tiempo ha cambiado lo suficiente para que hagan los cambios técnicos necesarios. Sobre los 5 años sacas una versión más nueva y a los 10 un coche nuevo por completo, esa actualización de media vida se hace para poder volver a tener el mismo número de ventas.

En informática pasa igual, hacemos versiones nuevas del sistema operativo, antes se hacía una versión nueva entre los 5 años y ahora se van haciendo actualizaciones cada cierto tiempo de la parte visual. Cuando te actualizan la parte visual lo que no tiene sentido es tener que reprogramar las actualizaciones, es decir, te da igual la parte visual. Lo que nos lleva es a tener

bibliotecas de componentes visuales donde el color cambia de forma o color pero que tenga la misma funcionalidad.

El principal problema es que cada sistema operativo tiene su propia interfaz. Tenemos que usar sistemas de librerías, de componentes, que son elementos visuales que tienen un comportamiento más o menos estables para poder meterlos en nuestras aplicaciones. Cuando se actualizan las librerías también las actualizaciones.

Tipos de interfaces:

- Nativas: dependen de donde se esté ejecutando el programa
- Hibridas: mezcla de nativo más web
- Web: son las interfaces de las páginas web

Las nativas son propias de cada sistema operativo. Cuando diseñas una aplicación de un sistema operativo especifico, usas la interfaz del sistema operativo al que corresponda porque si no el usuario lo rechaza.

Ejemplo: Puedo aceptar o no a una imagen que veo y se puede parecer poco o mucho a una imagen real. Los dibujos animados de los treinta son dibujos pintados, movimientos extraños, aparece Disney y hace que los dibujos hagan que te metas dentro de la acción, llega Pixar, son modelos 3D que se sabes que son modelos 3D, llegan a hacer películas 3D con personajes humanos, y no queda muy bien. Lo más abajo que está en la curva es un cadáver, después mejoran la técnica hasta que aceptas las cosas como si fuesen reales.

Los usuarios aceptan usar mi aplicación porque lo conocen. Cuando hago una aplicación nativa, estas en el pico más realista. Cuando haces una aplicación Web, 'no te trata de engañar', *Pixar*. Si el usuario lo acepta, se gana más dinero y por tanto, eso hace que haya libros y libros de como animar cosas para aprender a hacer animaciones que el usuario acepte.

Todo esto le ha sucedido a java durante mucho tiempo. Ha tenido 3 evoluciones visuales. La primera AWT, era fea y lenta. La segunda era SWING se sigue utilizando. La tercera es JavaFX que es la que se usa hoy en día.

Las interfaces web son un cambio son un cambio muy grande porque tienen una interfaz propia. En web todo se define con el DOM donde puedes definir las cosas como quieras.

Hace años, cada página iba a su bola. Hubo una disciplina que es la disciplina de usabilidad que creció sobre la época 2000'. Estudiaba que patrones de diseño visual funcionaba al usuario depende de que cosas necesitaban. Aparecieron unas guías de diseño. Apareció un FRAMEWORK visual que se llamaba BOOTSTRAP, viene de Twitter, se centraba en tener los elementos visuales adecuados para no tener que diseñar una web de 0.

Lo que sucede es que existe una separación en la percepción de usuario a lo largo del tiempo. El usuario cuando ve una aplicación con botones lo identifica con trabajo mientas que si es una web lo identifica con ocio. El usuario debe estar más inclinado a usar la web. Aunque son más ineficientes, les parece bonito y les gusta más. Llega un momento donde hay una separación muy grande entre las dos.

Una vez a alguien se le ocurrió mezclar las dos, y salen las hibridas. Las descargo, pero no quiero que parezca una aplicación sino una web, el tipo de lenguaje visual de una web.

Vaading.com

Es un framework que te permite hacer una web y luego la transformas en híbridos. La puedes programar directamente todo en Java.

Copilot es un nombre que representa el LMM de Microsoft. Vaadingcopilot está adaptado al framework de vaading. Vaading tiene un transpilador que va a generar el html.