**Taller**

**Santiago Salamanca Díaz**

**ADSO**

**Ficha 2558104**

**Institución: Servicio Nacional de Aprendizaje (SENA), CEET**

**Bogotá D.C.**

**1. A través de un mapa mental, explica la estrategia de manejo de datos.**

****

**2. Escoge una aplicación y explica cada una de las estrategias de obtención de datos.**

Una aplicación que podría ser utilizada para explicar las estrategias de obtención de datos es una aplicación de fitness. Esta aplicación recopila datos sobre el ejercicio físico realizado por los usuarios, como el número de pasos dados, la distancia recorrida, las calorías quemadas, etc. A continuación, se describen las diferentes estrategias de obtención de datos utilizadas en esta aplicación:

1. Entrada manual de datos: La aplicación permite a los usuarios introducir manualmente los datos de su actividad física. Por ejemplo, pueden ingresar el número de pasos que han dado en un día determinado.

2. Uso de sensores incorporados en el dispositivo: La aplicación aprovecha los sensores incorporados en el dispositivo móvil, como el acelerómetro y el giroscopio, para recopilar datos de movimiento. Estos sensores registran la aceleración y la orientación del dispositivo, lo que permite estimar los pasos dados y la distancia recorrida.

3. Integración con dispositivos wearables: La aplicación se puede conectar con dispositivos wearables, como pulseras de actividad o relojes inteligentes, que tienen sensores más precisos para medir la actividad física. Estos dispositivos proporcionan datos más precisos sobre los pasos, la distancia y las calorías quemadas.

4. Uso de servicios de localización: La aplicación utiliza los servicios de localización del dispositivo para rastrear la ubicación del usuario durante su ejercicio. Esto permite calcular la distancia recorrida con mayor precisión y proporcionar información sobre la ruta seguida.

5. Integración con otras aplicaciones y servicios: La aplicación puede integrarse con otras aplicaciones y servicios relacionados con la actividad física, como aplicaciones de seguimiento de nutrición o redes sociales de fitness. Esto permite obtener datos adicionales, como la ingesta calórica diaria o la actividad social del usuario.

Estas estrategias de obtención de datos permiten a la aplicación recopilar información precisa y detallada sobre la actividad física de los usuarios. Esta información puede utilizarse para ofrecer recomendaciones personalizadas, realizar un seguimiento del progreso y motivar a los usuarios a mantener un estilo de vida saludable.

**3. Que son los antipatrones de conectividad. Define y realiza un cuadro con las características mas importantes de cada antipatrón.**

Los "antipatrones de conectividad" son patrones de diseño o prácticas comunes que, en lugar de mejorar la conectividad y la comunicación en un sistema, tienden a dificultarla o generar problemas en la arquitectura. A menudo, estos antipatrones surgen de decisiones de diseño mal informadas o de falta de consideración de las mejores prácticas de conectividad. A continuación, se describen algunos de los antipatrones de conectividad más comunes y sus características principales:

1. Bola de Nieve (Big Ball of Mud):

Descripción: En este antipatrón, la arquitectura carece de estructura clara, y todas las partes del sistema se entrelazan de manera caótica. No hay una separación clara de módulos y responsabilidades.

Características:

Falta de estructura organizada.

Acoplamiento excesivo entre componentes.

Dificultad para entender y mantener el sistema.

2. Sobreconexión (Over-Connection):

Descripción: En este caso, hay un exceso de conexiones entre componentes o servicios en un sistema, lo que puede resultar en un alto tráfico de red y una mayor complejidad.

Características:

Exceso de comunicación y mensajes entre componentes.

Rendimiento deficiente debido a la sobrecarga de comunicación.

Mayor propensión a errores y problemas de escalabilidad.

3. Escalabilidad Vertical (Vertical Scalability):

Descripción: En lugar de permitir la escalabilidad horizontal, este antipatrón implica que los recursos se agregan verticalmente en un solo servidor, lo que limita la capacidad de escalabilidad y resiliencia.

Características:

Dependencia excesiva de recursos de servidor único.

Dificultad para manejar cargas de trabajo crecientes.

Falta de resiliencia y redundancia.

4. Red Intranable (Intricable Network):

Descripción: Este antipatrón se produce cuando la red del sistema es altamente compleja y difícil de entender. Las rutas de comunicación son confusas y poco claras.

Características:

Rutas de comunicación confusas y poco documentadas.

Dificultad para el mantenimiento y diagnóstico de problemas.

Riesgo de fallos y congestión en la red.

5. Múltiples Identidades (Multiple Identities):

Descripción: En este antipatrón, un componente o servicio tiene múltiples identidades o direcciones de red, lo que causa confusión y dificulta la comunicación.

Características:

Falta de una única fuente de verdad para la identidad de un componente.

Dificultad para enrutar y gestionar la comunicación.

Potencial para errores y problemas de seguridad.

Aquí tienes un cuadro comparativo con las características más importantes de cada uno de estos antipatrones:

|  |  |  |
| --- | --- | --- |
| Antipatrón | Descripción | Características Principales |
| Bola de Nieve | Falta de estructura organizada. | Acoplamiento excesivo, difícil de entender y mantener. |
| Sobreconexión | Exceso de comunicación entre componentes. | Rendimiento deficiente, mayor propensión a errores. |
| Escalabilidad Vertical | Dependencia de recursos de servidor único. | Dificultad para manejar cargas de trabajo crecientes. |
| Red Intranable | Rutas de comunicación confusas y poco claras. | Dificultad de mantenimiento y diagnóstico de problemas. |
| Múltiples Identidades | Múltiples identidades de un componente. | Dificultad en la gestión de comunicación y riesgo de errores. |

**4.Define: Manejo de datos, Datos instalados, Caché permanente, Caché temporal, Sin caché**

- Manejo de datos: Es el proceso de administrar y organizar los datos de forma eficiente para su almacenamiento, acceso, manipulación y análisis. Esto implica la creación, modificación, eliminación y consulta de datos, así como la implementación de políticas de seguridad y privacidad.

- Datos instalados: Son los datos que se encuentran almacenados en un dispositivo de manera permanente, ya sea en la memoria interna o en una tarjeta de almacenamiento. Estos datos suelen ser los archivos y aplicaciones que están instalados en el dispositivo y que pueden ser accedidos y utilizados por el usuario.

- Caché permanente: Es una forma de almacenamiento temporal de datos que se mantiene incluso después de reiniciar el dispositivo. En el contexto de software, se refiere a la información que se guarda en la memoria caché del dispositivo para acelerar la carga y acceso a aplicaciones o contenido específico. Esta caché permanente puede incluir imágenes, archivos multimedia, datos de configuración, entre otros.

- Caché temporal: Es una forma de almacenamiento temporal de datos que se utiliza para mejorar el rendimiento y la velocidad de acceso a determinados recursos. En este caso, los datos se guardan en la memoria caché del dispositivo por un periodo de tiempo limitado y se eliminan automáticamente después de cierto tiempo o cuando se necesita liberar espacio en la memoria. Esto permite reducir el tiempo de carga de aplicaciones o páginas web al acceder a datos que ya han sido previamente descargados y almacenados en la caché.

- Sin caché: Se refiere a la ausencia de almacenamiento en memoria caché de datos. En este caso, cada vez que se accede a un recurso o se realiza una consulta, se debe obtener los datos directamente desde la fuente original, lo que puede implicar un mayor tiempo de carga y un consumo de recursos más elevado.

5. **¿Explica porque es tan importante el rendimiento en las aplicaciones móviles? Justifica con bases tu respuesta.**

El rendimiento en las aplicaciones móviles es crucial debido a varios factores. En primer lugar, los usuarios tienen altas expectativas en cuanto a la velocidad y capacidad de respuesta de las aplicaciones. Si una aplicación es lenta o se congela con frecuencia, los usuarios se frustrarán y abandonarán rápidamente su uso, lo que puede resultar en una mala experiencia del usuario y una mala reputación para la aplicación y la empresa que la desarrolló.

Además, el rendimiento también está estrechamente relacionado con la eficiencia energética. Las aplicaciones móviles que consumen demasiada energía pueden agotar rápidamente la batería de un dispositivo, lo cual es especialmente problemático considerando que la mayoría de los usuarios dependen de sus teléfonos móviles durante todo el día. Una aplicación que consume demasiada energía puede ser percibida como poco confiable y poco práctica.

Otro aspecto importante del rendimiento en las aplicaciones móviles es la optimización del uso de los recursos del dispositivo. Los dispositivos móviles tienen recursos limitados, como memoria y capacidad de procesamiento. Una aplicación que consume demasiados recursos puede ralentizar el dispositivo en su conjunto y afectar negativamente la experiencia del usuario. Por lo tanto, es esencial que las aplicaciones estén bien optimizadas y utilicen eficientemente los recursos disponibles.

Además, el rendimiento también puede tener un impacto en la seguridad de una aplicación. Las aplicaciones que no están bien optimizadas pueden presentar vulnerabilidades que podrían ser aprovechadas por los hackers. Por lo tanto, es importante que las aplicaciones móviles sean rápidas y eficientes para minimizar los riesgos de seguridad.

7. **Define: Performance, Memory bloat, Error de no dormir, Error de inmortalidad, Error de bucle/del circuito de energía, Usos de API codiciosos de energía, ANR, Retrasos en la GUI, Renderizado lento**

- Performance: Es la capacidad de un sistema o software para ejecutar tareas de manera eficiente y rápida, cumpliendo con los requisitos de tiempo y recursos establecidos.

- Memory bloat: Se refiere al aumento excesivo del consumo de memoria en un programa o sistema, generalmente debido a la asignación ineficiente o el mal manejo de la memoria.

- Error de no dormir: Se refiere a un error en el cual un programa o sistema no se detiene o entra en un estado de espera cuando debería hacerlo, lo que puede llevar a un mal funcionamiento o bloqueo.

- Error de inmortalidad: Se refiere a un error en el cual un programa o sistema no se detiene o finaliza correctamente, lo que puede provocar un consumo excesivo de recursos y una degradación del rendimiento.

- Error de bucle/del circuito de energía: Se refiere a un error en el que un programa o sistema entra en un bucle infinito o se queda atrapado en un circuito de ejecución continua, lo que puede provocar un consumo excesivo de energía y llevar a un mal funcionamiento.

- Usos de API codiciosos de energía: Se refiere al uso excesivo de las API (Application Programming Interface) de un sistema o software, lo que puede resultar en un consumo excesivo de energía y reducir la duración de la batería de un dispositivo.

- ANR (Application Not Responding): Se refiere a un estado en el que una aplicación no responde a las interacciones del usuario durante un período prolongado de tiempo, lo que puede causar frustración al usuario y afectar negativamente la experiencia de uso.

- Retrasos en la GUI: Se refiere a la demora o lentitud en la respuesta de la interfaz gráfica de usuario (GUI) de un programa o sistema, lo que puede causar una experiencia de usuario insatisfactoria.

- Renderizado lento: Se refiere a la velocidad lenta en la cual se muestra o actualiza el contenido visual en la pantalla de un programa o sistema, lo que puede afectar la fluidez y la experiencia de usuario.

**8. Realiza un cuadro comparativo entre GC (Recolector de basura) y ARC (Conteo Automático de Referencias).**

|  |  |  |
| --- | --- | --- |
| Característica | GC (Recolector de Basura) | ARC (Conteo Automático de Referencias) |
| Definición | GC es un mecanismo que automáticamente identifica y recoge objetos no utilizados en el heap de memoria, liberando recursos. | ARC es un mecanismo que realiza un seguimiento automático de referencias a objetos y libera la memoria cuando ya no hay referencias activas a ellos. |
| Uso Principal | Comúnmente utilizado en lenguajes de programación como Java y C#. | Comúnmente utilizado en lenguajes de programación como Swift y Objective-C. |
| Control | El programador no tiene control directo sobre cuándo se realiza la recolección de basura. | El programador no tiene control directo sobre cuándo se liberan los objetos. |
| Rendimiento | Puede causar breves pausas en la ejecución del programa cuando se ejecuta la recolección de basura. | Tiende a ser más eficiente en términos de rendimiento, ya que no implica pausas significativas en la ejecución. |
| Ciclo de Vida del Objeto | Los objetos no se liberan inmediatamente después de que se vuelven inaccesibles. | Los objetos se liberan tan pronto como ya no hay referencias activas a ellos. |
| Identificación de Fugas de Memoria | Puede ayudar a evitar fugas de memoria identificando y liberando objetos no utilizados automáticamente. | Puede ayudar a evitar fugas de memoria identificando referencias fuertes cicladas automáticamente. |
| Programación Concurrente | Puede requerir estrategias de programación concurrente y sincronización para evitar problemas de acceso a objetos en el heap. | Menos susceptible a problemas de concurrencia, ya que no requiere gestión manual de bloqueo o sincronización. |

**9. ¿Qué es una referencia fuerte y débil? ¿En qué radica su importancia?**

Una referencia fuerte es un tipo de referencia que mantiene un objeto en memoria mientras exista una referencia activa a él. En otras palabras, mientras haya al menos una referencia fuerte apuntando a un objeto, este no será recolectado por el recolector de basura.

Por otro lado, una referencia débil es un tipo de referencia que permite que el objeto sea recolectado por el recolector de basura si no hay otras referencias fuertes que lo mantengan en memoria. Una referencia débil es útil en situaciones donde se necesita acceder a un objeto mientras esté disponible, pero no es crítico mantenerlo en memoria.

La importancia de las referencias fuertes y débiles radica en la gestión eficiente de la memoria en un programa. El uso adecuado de estos tipos de referencias puede ayudar a evitar problemas de fugas de memoria y mejorar el rendimiento de la aplicación. Además, las referencias débiles son especialmente útiles en casos donde se necesita una caché temporal o en situaciones donde se trabaja con objetos que pueden ser creados y eliminados frecuentemente.

**10. Realiza un mapa mental del capítulo 12 del libro en estudio.**