Unidad Didáctica 5:
CÓMO HACER QUE TUS APPS SEAN RÁPIDAS Y LIVIANAS

Programación Dirigida por Eventos

UAX Universidad Alfonso X el Sabio

ÍNDICE

Contenido

1. Introducción

¡Bienvenido a la asignatura Programación Dirigida por Eventos!

En esta unidad didáctica, nos enfocaremos en "Optimización del rendimiento" en aplicaciones Android. La optimización es un aspecto crucial del desarrollo de software que asegura que las aplicaciones sean rápidas, eficientes y ofrezcan una experiencia de usuario fluida. Aprenderás a identificar y solucionar problemas de rendimiento, a utilizar herramientas de perfilado y a implementar técnicas para mejorar la eficiencia de la memoria, el uso de la CPU y la batería.

Ten en cuenta:

Es esencial para los desarrolladores no solo crear aplicaciones funcionales, sino también asegurarse de que estas aplicaciones funcionen de manera eficiente en una variedad de dispositivos con diferentes capacidades de hardware. La optimización del rendimiento impacta directamente en la satisfacción del usuario y en la percepción de calidad de la aplicación.

Temas que se tratarán en esta unidad:

1. **Herramientas de perfilado en Android Studio:** Uso de Android Profiler para monitorear el rendimiento de la CPU, memoria, red y batería.
2. **Identificación de cuellos de botella:** Técnicas para detectar y solucionar problemas que afectan el rendimiento.
3. **Optimización de la memoria:** Métodos para reducir el consumo de memoria y evitar fugas de memoria.
4. **Optimización de la CPU:** Técnicas para asegurar que las operaciones intensivas en CPU se manejen eficientemente.
5. **Optimización de la red:** Estrategias para reducir el uso de datos y mejorar la velocidad de transferencia.
6. **Optimización del uso de la batería:** Implementación de técnicas para reducir el consumo de batería y prolongar la vida útil del dispositivo.
7. **Reducción del tiempo de carga de la aplicación:** Métodos para mejorar el tiempo de inicio y la velocidad de carga de las diferentes pantallas y funciones dentro de la aplicación.
8. **Uso eficiente de recursos gráficos:** Técnicas para manejar y renderizar gráficos de manera eficiente, minimizando el overdraw y optimizando el uso de GPU.

Al finalizar esta unidad, deberías ser capaz de aplicar técnicas de optimización de rendimiento a tus aplicaciones Android, asegurando que sean rápidas, eficientes y proporcionen una experiencia de usuario óptima.

2. Objetivos

Objetivo general:

- Proporcionar a los estudiantes las habilidades necesarias para optimizar el rendimiento de aplicaciones Android, mejorando la eficiencia de la memoria, CPU, red y batería.

Objetivos específicos:

En esta unidad se establecen tres objetivos específicos:

1. **Comprender y utilizar herramientas de perfilado en Android Studio:** Aprender a identificar y solucionar problemas de rendimiento utilizando Android Profiler y otras herramientas de perfilado.
2. **Implementar técnicas de optimización de recursos:** Desarrollar la capacidad de aplicar técnicas para optimizar el uso de memoria, CPU, red y batería en aplicaciones Android.
3. **Mejorar la experiencia del usuario mediante la optimización del rendimiento:** Aplicar estrategias para reducir el tiempo de carga, mejorar la velocidad de las operaciones y asegurar una experiencia de usuario fluida y eficiente.

3. Localización

3.1. Localización en Android

Visión General de la Elección de Idioma y Configuración Regional del Usuario

Configuración de Idioma y Configuración Regional

Los usuarios eligen su idioma y configuración regional en la configuración del dispositivo. Ejemplos de la misma lengua en diferentes configuraciones regionales incluyen:

- Inglés (EE.UU.) vs. Inglés (Reino Unido)
- Francés (Francia) vs. Francés (Canadá)

Desde Android Nougat (API 24) y versiones más recientes, los usuarios pueden elegir múltiples idiomas en orden de preferencia.

Modificación de Aplicaciones para Diferentes Idiomas

Para llegar a más usuarios, es importante localizar tu aplicación para cada configuración regional. Esto implica:

- Añadir recursos de idioma/configuración regional para el texto traducido.
- Ajustar el diseño para soportar diferentes idiomas.
- La aplicación mostrará el idioma predeterminado para configuraciones regionales no soportadas.

Preparación de una Aplicación para la Localización

Pasos para Preparar una Aplicación

1. **Identificar Todos los Elementos a Traducir**: Todo el texto de la interfaz de usuario, incluyendo menús de contenido, pestañas de navegación, mensajes, encabezados y etiquetas de campos de entrada.
2. **Extraer Todas las Cadenas como Recursos en strings.xml**: Centralizar las cadenas de texto para facilitar la traducción.
3. **Usar el Editor de Traducciones en Android Studio**: Añadir idiomas y traducciones de manera eficiente.
4. **Cambiar Diseños para Idiomas RTL**: Asegurar que el diseño soporte idiomas de derecha a izquierda.

Identificación de Elementos a Traducir

Todos los elementos de texto de la UI y aquellos que requieren formato de configuración regional, como fechas, horas, números y monedas, deben ser identificados y preparados para la traducción.

Mejores Prácticas para la Localización de Texto

- Añadir idiomas usando el editor de traducciones.
- Proveer traducciones para todas las cadenas.
- No concatenar piezas para formar oraciones o frases.
- Usar cadenas de formato con String.format() para cadenas con contenido variable.

Consejos para el Diseño y los Medios

- Evitar imágenes y colores que puedan ofender a ciertas culturas.
- Evitar el humor, ya que rara vez se traduce bien.
- No usar imágenes que contengan texto.
- Incluir comentarios para el traductor en strings.xml.

Uso del Editor de Traducciones en Android Studio

Añadir Idiomas y Traducciones

1. **Abrir el Editor de Traducciones**: En la parte superior de strings.xml en el editor de diseño.
2. **Añadir un Idioma**: Hacer clic en el botón del globo para añadir un idioma.
3. **Ingresar Traducciones**: Seleccionar una celda en la columna del idioma y marcar la casilla "Untranslatable" para cualquier clave que no deba ser traducida. Para las claves que deben ser traducidas, ingresar la traducción en el campo de traducción.

Ajuste de Diseños para Idiomas RTL

Mirroring de Diseño

El mirroring de diseño (API 17+) redibuja el diseño para idiomas RTL. Para habilitar el mirroring en AndroidManifest.xml:

```
<application
   android:supportsRtl="true">
```

Compatibilidad de Mirroring de Diseño

Para minSdkVersion < 17, añade atributos "start" y "end" además de "left" y "right". Para minSdkVersion >= 17, reemplaza los atributos "left" y "right" con "start" y "end".

Ajuste de Posicionamiento Izquierda/Derecha

Añadir o reemplazar "left" y "right" por "start" y "end" para:

- **Atributos de Constraint**: app:layout_constraintLeft_toRightOf="@+id/product_image" con app:layout_constraintStart_toEndOf="@+id/product_image"
- **Atributos de RelativeLayout**: android:layout_alignParentLeft con android:layout_alignParentStart
- **Márgenes y Padding**

Pruebas de Aplicaciones con Diferentes Idiomas y Configuraciones Regionales

Vista Previa del Mirroring de Diseño

Para previsualizar el mirroring de diseño para RTL en el editor de diseño:

1. Hacer clic en la pestaña **Design**.
2. Seleccionar el idioma en el menú de idiomas.

Uso de Configuración en Marshmallow y Anteriores

En un dispositivo o emulador, selecciona:

- **Languages & input > Language**
- Seleccionar un idioma y configuración regional.

Uso de Configuración en Nougat y Nuevas

En un dispositivo o emulador, selecciona:

- **Languages & input > Languages**
- Seleccionar **Add a language** y elegir el idioma y configuración regional.
- Mover el idioma en la lista de preferencias.

Ejecución de la Aplicación

Cambia el idioma en el dispositivo o emulador y ejecuta la aplicación para verificar la correcta localización.

3.2. Localización y Configuraciones Regionales en Android

Visión General de la Personalización de la Configuración Regional

Usando la Configuración Regional Elegida por el Usuario

Los usuarios seleccionan su idioma y configuración regional en la configuración del dispositivo. Esto incluye no solo el idioma sino también el formato de fechas, horas, números y monedas. Ejemplos de la misma lengua en diferentes configuraciones regionales incluyen:

- Inglés (EE.UU.) vs. Inglés (Reino Unido)
- Francés (Francia) vs. Francés (Canadá)

Desde Android Nougat (API 24) y versiones más recientes, los usuarios pueden elegir múltiples idiomas en orden de preferencia, permitiendo una experiencia más personalizada.

Mejores Prácticas para la Configuración Regional

Para asegurar que tu aplicación se adapte correctamente a las preferencias del usuario, sigue estas mejores prácticas:

- **Convertir formatos de fecha, hora, número y moneda a la configuración regional del usuario**: Adaptar estos elementos para que sean coherentes con las convenciones locales del usuario.
- **Usar métodos de clase para convertir, no codificar los formatos**: Utilizar las herramientas proporcionadas por Android para asegurar la coherencia y precisión, evitando errores y inconsistencias.
- **Almacenar datos en el formato de la configuración regional predeterminada y convertir a la del usuario según sea necesario**: Facilitar el mantenimiento y la coherencia de los datos, asegurando que siempre se presenten de la manera más apropiada para el usuario.

Formateo de Fechas y Horas

Clase DateFormat

La clase DateFormat se utiliza para aplicar el formato de la configuración regional elegida por el usuario. Esto permite que las fechas y horas se muestren de acuerdo con las convenciones locales.

Métodos de DateFormat

- **getDateInstance()**: Obtiene el formato de fecha para la configuración regional del usuario.
- **getDateInstance(int style, Locale aLocale)**: Obtiene el formato de fecha para la configuración regional proporcionada.
- **getTimeInstance()**: Obtiene el formato de hora para la configuración regional del usuario.
- **getDateTimeInstance()**: Obtiene el formato combinado de fecha y hora para la configuración regional del usuario.

Ejemplo de Uso

```
final Date myDate = new Date();
String myFormattedDate = DateFormat.getDateInstance().format(myDate);
TextView dateView = (TextView) findViewById(R.id.date);
dateView.setText(myFormattedDate);
```

Este ejemplo muestra cómo formatear una fecha según la configuración regional del usuario y asignarla a un TextView.

Importancia del Formateo Correcto de Fechas y Horas

El uso correcto del formato de fechas y horas no solo mejora la experiencia del usuario, sino que también evita confusiones y errores en la interpretación de la información.

Formateo de Números

Clase NumberFormat

La clase NumberFormat se utiliza para formatear números para cualquier configuración regional. Android proporciona constantes de configuración regional para muchos países y regiones, facilitando la personalización de la aplicación.

Métodos de NumberFormat

- **getInstance()**: Obtiene el formato de número para la configuración regional actual.
- **getIntegerInstance()**: Retorna un formato de entero para la configuración regional actual.
- **getPercentInstance()**: Retorna un formato de porcentaje para la configuración regional actual.
- **getCurrencyInstance()**: Retorna un formato de moneda para la configuración regional actual.

Ejemplo de Uso

NumberFormat numberFormat = NumberFormat.getInstance(Locale.FRANCE);

Este ejemplo muestra cómo obtener una instancia de NumberFormat para la configuración regional de Francia.

Parseo de Cadenas a Números

El parseo es el proceso de convertir una cadena de texto en un número. Esto es útil cuando se necesita procesar entradas de usuario.

Ejemplo de Uso

myQuantity = numberFormat.parse(qtyInput.getText().toString()).intValue();

Este ejemplo muestra cómo convertir una entrada de texto en un número entero utilizando NumberFormat.

Importancia del Formateo Correcto de Números

El formateo adecuado de números asegura que los datos se presenten de manera comprensible y precisa, respetando las convenciones locales del usuario.

Formateo de Monedas

Uso de NumberFormat para Monedas

Utiliza NumberFormat para los formatos de moneda. Esto permite que los precios y otros valores monetarios se muestren de acuerdo con la configuración regional del usuario, incluyendo el símbolo de moneda adecuado y el formato correcto.

Ejemplo de Uso

```
NumberFormat currencyFormat = NumberFormat.getCurrencyInstance();
String myFormattedPrice = currencyFormat.format(myPrice);
TextView localePrice = (TextView) findViewById(R.id.price);
localePrice.setText(myFormattedPrice);
```

Este ejemplo muestra cómo formatear un precio según la configuración regional del usuario y asignarlo a un TextView.

Importancia del Formateo Correcto de Monedas

El formateo correcto de monedas es crucial para aplicaciones que manejan transacciones financieras, ya que asegura que los usuarios comprendan claramente los valores y costos presentados.

Recuperación y Uso de la Configuración Regional Elegida por el Usuario

Obtener el Código de País/Región

Utiliza Locale.getDefault() para obtener la configuración regional actual y Locale.getDefault().getCountry() para obtener el código de país/región. Esto permite que la aplicación adapte su contenido y comportamiento según la ubicación del usuario.

Ejemplo de Uso

```
String deviceLocale = Locale.getDefault().getCountry();
if (deviceLocale.equals("FR") || deviceLocale.equals("IL")) {
   // Si la configuración regional es Francia o Israel...
}
```

Uso de Constantes de Locale

Usa constantes de Locale para crear objetos Locale. Esto facilita la creación y manejo de configuraciones regionales específicas.

Ejemplo de Uso

```
currencyFormat = NumberFormat.getCurrencyInstance(Locale.US);
```

Este ejemplo muestra cómo obtener un formato de moneda para la configuración regional de Estados Unidos.

Importancia de la Recuperación Correcta de la Configuración Regional

Asegurar que la aplicación recupere y utilice correctamente la configuración regional del usuario mejora la relevancia y usabilidad de la aplicación.

Adición de Recursos para Diferentes Configuraciones Regionales

Directorios de Recursos en el Proyecto

Los recursos se organizan en directorios específicos para diferentes idiomas y configuraciones regionales. Estos incluyen:

- **values**: Colores de texto, dimensiones y estilos para el idioma/configuración regional predeterminada.
- **drawable**: Drawables para el idioma/configuración regional predeterminada.
- **layout**: Diseños para el idioma/configuración regional predeterminada.

Gestión de Directorios de Recursos

Android Studio proporciona directorios de recursos predeterminados en el proyecto. Usa Android Studio para añadir un directorio de recursos para cada idioma/configuración regional soportada. Android selecciona el directorio de recursos que mejor coincide con el idioma/configuración regional elegida por el usuario.

Importancia de los Valores Predeterminados

Si el usuario selecciona un idioma/configuración regional no soportada por tu aplicación, Android elige los recursos predeterminados. Asegúrate de que la aplicación incluya un conjunto completo de recursos predeterminados para evitar problemas de presentación o funcionalidad.

Añadir Directorios de Recursos

1. Haz clic derecho en res y selecciona New > Android Resource Directory.
2. Elige el tipo de recurso (values, drawable, etc.).
3. Selecciona la configuración regional y haz clic en >> para seleccionar un idioma y configuración regional.

Formato de Nombre para Directorios de Recursos

- <resource>-<language>[-r<country>]
- **<resource>**: Tipo de recurso (values, drawable, etc.)
- **<language>**: Idioma (en para inglés, fr para francés)
- **<country>**: Código opcional (US para EE.UU., FR para Francia)

Ejemplos

- values-fr-rFR: Valores para francés en Francia
- drawable-fr-rFR: Drawables para francés en Francia

Organización y Manejo de Recursos

Es importante mantener una estructura clara y organizada para los recursos de diferentes configuraciones regionales. Esto facilita el mantenimiento y la actualización de la aplicación.

4. Accesibilidad

4.1. Accesibilidad en Android

Acerca de la Accesibilidad

Objetivo de la Accesibilidad

El objetivo de la accesibilidad es permitir que todos los usuarios, incluidas las personas con discapacidades, puedan usar tus aplicaciones. El framework de Android proporciona diversas características de accesibilidad que puedes integrar en tus aplicaciones para asegurar su usabilidad por parte de todos los usuarios.

Características de Accesibilidad en la Configuración de Android

En **Android > Configuración > Accesibilidad**, puedes encontrar varias opciones:

- **Google TalkBack**: Interactúa utilizando retroalimentación táctil y hablada.
- **Seleccionar para hablar**: Proporciona retroalimentación hablada solo en ciertos momentos.
- **Acceso con interruptores**: Alternativa al uso de la pantalla táctil (utilizando interruptores o teclado).
- **Aplicación Voice Access**: Controla el dispositivo con comandos de voz (solo en inglés).
- **Google BrailleBack**: Conecta una pantalla braille actualizable a través de Bluetooth (disponible en Google Play).
- **Ajustar el tamaño de la pantalla o la fuente**.
- **Usar gestos de magnificación para hacer zoom temporalmente**.
- **Usar texto de alto contraste, inversión de colores y corrección de color**.
- **Activar subtítulos y especificar opciones de subtítulos cerrados**.

Diseño y Organización de la Aplicación

Directrices para el Diseño

1. **Evitar Diseños Complejos y Pantallas Desordenadas**
   - Los usuarios con visión reducida pueden tener dificultades.
   - Demasiadas vistas dificultan la navegación con un lector de pantalla.
2. **Colocar los Elementos Según su Nivel de Importancia**
   - Las acciones importantes deben estar en la parte superior o inferior.
   - Asegúrate de que los elementos clave se destaquen.
3. **Usar las Directrices de Accesibilidad de Material Design**
   - Agrupar elementos relacionados juntos en proximidad.

- Diseñar utilizando las guías de usabilidad de Material Design para barras de herramientas, menús, botones de acción, etc.

Objetivos Táctiles

Los objetivos táctiles son las partes de la pantalla que responden a la entrada del usuario. Deben extenderse más allá de los límites visuales del elemento y tener al menos 48 x 48 dp para facilitar su interacción.

Ejemplo de Objetivo Táctil

Un icono puede ser de 24 x 24 dp, pero el objetivo táctil debería ser de 48 x 48 dp incluyendo el padding.

Etiquetas y Descripciones de Contenido

Descripciones de Contenido

Las vistas que no contienen texto necesitan descripciones para que los lectores de pantalla puedan proporcionar información sobre su propósito.

Cuándo Añadir Descripciones de Contenido

- **ImageView y ImageButton** que no tienen texto.
- **Botones** cuyo texto no describe completamente su acción.
- **Vistas Estáticas**: Usar android:contentDescription en el diseño.
- **Vistas Dinámicas**: Usar setContentDescription() en tiempo de ejecución.

Ejemplo de Adición de Descripción de Contenido

```
<ImageView
  android:id="@+id/image_partly_cloudy"
  android:layout_width="200dp"
  android:layout_height="160dp"
  app:srcCompat="@drawable/partly_cloudy"
  android:contentDescription="@string/partly_cloudy" />
```

Creación de Descripciones Útiles

- Usar verbos de acción para describir lo que hace el elemento de la UI.
- No incluir el nombre del elemento de la UI (como "botón").
- No indicar a los usuarios cómo interactuar con el control, ya que pueden estar navegando por teclado o voz.

- Asegurar que las descripciones sean únicas, especialmente en vistas colectivas como RecyclerView.
- Mantener la misma descripción para la misma vista en todas partes.

Mejores Prácticas para el Diseño de la UI

Herramientas de Diseño de UI para la Accesibilidad

- **Paletas de Colores y Herramientas de Color de Material Design**: Ajustar la paleta de colores y verificar la legibilidad.
- **Contraste**: Asegurar que los colores tengan suficiente contraste entre los elementos.

Recomendaciones de Contraste

El contraste recomendado por W3C es:

- **Texto Pequeño**: Al menos 4.5:1 contra el fondo.
- **Texto Grande**: Al menos 3:1 contra el fondo.
- **Logos**: No necesitan cumplir con los mínimos de contraste.

Ejemplo de Buen y Mal Contraste

Un título con una proporción de 1.42:1 tiene bajo contraste, mientras que una proporción de 4.61:1 es adecuada.

Pistas Visuales

- Usar diferentes formas o tamaños para ayudar a los usuarios con deficiencias de visión de color.
- Proveer patrones visuales o pistas de texto.
- Añadir retroalimentación táctil o auditiva.

Audio y Video

Accesibilidad de Audio y Video

Para asegurar que las personas con dificultades auditivas puedan acceder a los contenidos:

- Proveer alternativas visuales para el audio.
- Proveer alternativas auditivas para los visuales.
- No usar solo retroalimentación de audio.

Ejemplo de Alternativas

Para retroalimentación y alertas de audio esenciales, proporcionar subtítulos cerrados, transcripciones u otras alternativas visuales.

Pruebas de Accesibilidad

Herramientas para Probar la Accesibilidad

1. **Google TalkBack**: Realizar pruebas manuales para retroalimentación táctil y hablada.
2. **Accessibility Scanner**: Proporciona sugerencias de mejora.
3. **Frameworks de Prueba**: Espresso y Robolectric incluyen verificaciones de accesibilidad.
4. **Lint de Android Studio**: Muestra advertencias de accesibilidad y enlaces al código fuente correspondiente.

Ejemplo de Prueba con Google TalkBack

- Activa TalkBack en **Configuración > Accesibilidad > TalkBack**.
- Explora la aplicación tocando y escuchando la retroalimentación.
- Verifica que todas las vistas y actividades sean accesibles.

Uso de Accessibility Scanner

- Instalar desde Google Play.
- Activar en **Configuración > Accesibilidad > Accessibility Scanner**.
- Navegar por la aplicación y recibir notas sobre posibles mejoras.

5. Rendimiento

5.1 Rendimiento en Android

¿Qué es un Buen Rendimiento?

Características de un Buen Rendimiento

Un rendimiento óptimo en una aplicación se caracteriza por:

- **Carga rápida**: La aplicación debe iniciar rápidamente, idealmente en menos de 2 segundos. Esto incluye tanto la carga inicial de la aplicación como la carga de las diferentes pantallas y funciones dentro de la aplicación.
- **Animaciones fluidas**: Las animaciones deben ser suaves y sin interrupciones, manteniendo una tasa constante de 60 fotogramas por segundo (FPS) para evitar saltos y lag.
- **Sin fallos**: La aplicación no debe fallar inesperadamente. La estabilidad es crucial para mantener la confianza y satisfacción del usuario.
- **Respeto a la batería**: La aplicación debe utilizar eficientemente los recursos de la batería del dispositivo, evitando procesos innecesarios y optimizando las tareas en segundo plano.
- **Optimización de la transmisión de datos**: La aplicación debe manejar los datos de internet de manera eficiente, minimizando el uso de datos y siendo consciente de las limitaciones del plan de datos del usuario.
- **Respuesta consistente y rápida**: Las acciones del usuario deben recibir respuestas rápidas y consistentes, garantizando una experiencia de usuario fluida y sin demoras perceptibles.

Consecuencias del Mal Rendimiento

Un mal rendimiento puede manifestarse como:

- **La aplicación no carga o tarda mucho en cargar**: Esto puede frustrar a los usuarios y hacer que abandonen la aplicación antes de siquiera interactuar con ella.
- **Las animaciones se interrumpen**: Las animaciones que no son fluidas pueden dar una impresión de baja calidad y falta de pulido.
- **La aplicación se bloquea de manera aleatoria**: Los fallos inesperados pueden llevar a la pérdida de datos y a una experiencia de usuario negativa.
- **La batería se drena rápidamente**: Una aplicación que consume demasiada batería puede ser desinstalada rápidamente por los usuarios.
- **La aplicación consume muchos datos del plan de datos**: Los usuarios pueden desactivar el uso de datos en segundo plano o desinstalar la aplicación si consume demasiados datos.
- **La respuesta a las acciones del usuario es lenta o inconsistente**: Esto puede hacer que la aplicación sea frustrante de usar y que los usuarios busquen alternativas.

Importancia del Rendimiento

Las aplicaciones con buen rendimiento reciben mejores calificaciones y son menos propensas a ser desinstaladas. Para destacar, las aplicaciones deben:

- **Tener un tamaño de APK pequeño**: Un archivo APK más pequeño se descarga más rápido y ocupa menos espacio en el dispositivo del usuario.
- **Ejecutarse de manera rápida y fluida**: Una ejecución fluida mejora la percepción de calidad y profesionalismo de la aplicación.
- **Utilizar eficientemente los recursos de memoria, almacenamiento del dispositivo y batería**: Una aplicación optimizada no solo mejora la experiencia del usuario sino que también puede funcionar en una mayor variedad de dispositivos.
- **Ser consideradas con el plan de datos al transferir datos por internet**: Minimizar el uso de datos es crucial para usuarios con planes de datos limitados.

---

16 Milisegundos por Fotograma

Tasa de Refresco y Fotogramas por Segundo (FPS)

El ojo humano percibe movimiento suave cuando las imágenes se muestran en rápida sucesión. La calidad se mide en fotogramas por segundo (FPS):

- **10-12 páginas/segundo**: Flip book. La animación es perceptible pero no muy fluida.
- **24-30 FPS**: Películas. Este es el estándar para la mayoría de las producciones cinematográficas, proporcionando una experiencia visual fluida.
- **60 FPS**: Alta calidad. Este estándar es común en videojuegos y aplicaciones móviles de alta calidad, proporcionando una experiencia extremadamente suave.

La mayoría del hardware moderno de dispositivos refresca la pantalla a 60 FPS, lo que equivale a 16.6 milisegundos por fotograma. El software y el sistema deben coincidir con esa tasa para dibujar un fotograma completo cada 16.6 ms. La aplicación no obtiene todo ese tiempo, ya que el sistema Android también necesita algunos recursos.

Dropped Frames (Fotogramas Caídos)

Si la aplicación no termina de actualizar en 16 ms, el fotograma se pierde, lo que causa interrupciones visibles para el usuario. Esto se percibe como tartamudeo o "stutter" en la animación, y puede deteriorar significativamente la experiencia del usuario.

Prueba Básica de Rendimiento

Pasos para una Prueba Básica de Rendimiento

1. **Instalar en el dispositivo objetivo de gama baja**: Probar la aplicación en el dispositivo de menor capacidad que se espera que la ejecute, para asegurarse de que funcione bien en todas las condiciones.
2. **Usar la aplicación exhaustivamente** y tomar notas detalladas: Probar todas las funcionalidades y documentar cualquier problema o ralentización.
3. **Intentar hacer que la aplicación falle**: Forzar situaciones límite para asegurar que la aplicación maneje bien los errores.
4. **Pedir a un amigo que la pruebe**: Obtener una perspectiva diferente puede revelar problemas que el desarrollador no ha notado.
5. **Realizar una mini prueba de usabilidad**: Observar cómo otros usuarios interactúan con la aplicación puede proporcionar información valiosa sobre la experiencia del usuario.

Dispositivo Físico vs. Emulador

Los datos de rendimiento en el emulador son menos precisos. Puede ser difícil probar con redes de baja velocidad y poco fiables en dispositivos físicos, pero se puede simular usando el emulador y ajustando sus configuraciones.

Verificación de la Tasa de Fotogramas

Perfilado de Renderización de GPU

La herramienta de perfilado de renderización de GPU visualiza cuánto tiempo toma una aplicación para dibujar fotogramas.

Iniciar el Perfilado de Renderización de GPU

1. Ir a **Configuración > Opciones de desarrollador**.
2. Desplazarse a la sección de **Monitoreo** y seleccionar **Perfilado de renderización de GPU**.
3. Elegir **En pantalla como barras**.

Interpretación de las Barras de Perfilado

- **Una barra vertical = un fotograma**.
- **Colores = etapas de renderización**.
- **Línea horizontal verde = 16 ms por fotograma**.
- Si la barra se extiende por encima de la línea, el fotograma es mayor a 16 ms.
- Si la barra está por debajo de la línea, el fotograma es menor a 16 ms.

Utilidad del Perfilado de GPU

- **Línea horizontal verde**: Ver cómo los fotogramas se desempeñan respecto al objetivo de 16 ms por fotograma.
- **Barras**: Identificar si alguna parte del pipeline de renderización se destaca.
- Buscar picos en el tiempo de renderización asociados a acciones del usuario o del programa.

Abordar Problemas de Rendimiento

Uso de Herramientas de Perfilado de Rendimiento

En el Dispositivo

- **Profile GPU Rendering Tool**: Visualiza cómo la GPU está manejando el renderizado de los gráficos de la aplicación.
- **Debug GPU Overdraw Tool**: Ayuda a identificar áreas donde la aplicación está dibujando más de lo necesario.

En Android Studio

- **Layout Inspector**: Permite inspeccionar y depurar el diseño de la aplicación.
- **Android Profiler**: Proporciona datos en tiempo real sobre CPU, memoria, red y uso de energía.
- **Systrace y dumpsys**: Ofrecen un análisis detallado del rendimiento del sistema.
- **Batterystats y Battery Historian**: Ayudan a identificar y resolver problemas relacionados con el consumo de batería.

Ciclo de Mejora del Rendimiento

1. **Recopilar información**: Usar herramientas para recolectar datos detallados y variados en múltiples dispositivos y patrones de uso.
2. **Analizar para obtener información**: Entender los datos en el contexto de tu aplicación. Pueden haber múltiples problemas, y el problema más obvio puede enmascarar problemas más profundos.
3. **Tomar acción y solucionar problemas**: Evaluar múltiples formas de resolver problemas considerando restricciones y compensaciones. Esto incluye la gestión de recursos, presupuestos y plazos.
4. **Verificar e iterar**: Recolectar nuevos datos después de los cambios y comparar con los datos originales para asegurar que los problemas se hayan resuelto y determinar si se necesitan más mejoras.

Mantener tu Aplicación Receptiva

Monitoreo con Android Profiler

Monitorear mientras se desarrolla la aplicación proporciona datos en tiempo real del dispositivo sobre:

- **CPU**: Analiza el uso del procesador.
- **Memoria**: Monitorea el uso de la memoria para evitar fugas y excesos.
- **Red**: Evalúa el tráfico de datos para optimizar las transferencias.
- **Asignaciones**: Verifica la eficiencia en la gestión de objetos y memoria.

El uso continuo de estas herramientas durante el desarrollo asegura que la aplicación se mantenga receptiva y eficiente.

5.2 Renderizado y Diseño en Android

Fundamentos del Renderizado

Renderizado y Hardware

El proceso de renderizado implica varias etapas y componentes de hardware:

- **CPU**: Calcula las listas de visualización y los comandos gráficos que definen un fotograma de salida.
- **GPU**: La unidad de procesamiento gráfico se encarga de renderizar a la pantalla, aliviando la carga de la CPU en aplicaciones intensivas en gráficos.
- **Memoria**: Almacena imágenes y datos necesarios para el renderizado.
- **Batería**: Proporciona energía eléctrica para el hardware.

Lista de Visualización

Una lista de visualización define la imagen de salida usando primitivas como líneas y formas básicas. Esta lista se crea a partir de las instrucciones del programa en la CPU y se pasa a la GPU para ejecutar el renderizado en la pantalla.

Pipeline de Renderizado

El pipeline de renderizado es un modelo conceptual que describe los pasos para renderizar información gráfica en la pantalla:

1. **Creación de la lista de visualización**: A partir de las instrucciones del programa en la CPU.

2. **Paso de la lista de visualización a la GPU**.
3. **Ejecución de la lista de visualización por la GPU** para dibujar en la pantalla.

Restricciones del Hardware

El hardware juega un papel crucial en el rendimiento. Superar las limitaciones del hardware puede:

- Hacer que las aplicaciones sean lentas.
- Causar un rendimiento deficiente de la pantalla.
- Agotar la batería rápidamente.

Minimización del Overdraw

¿Qué es el Overdraw?

El overdraw ocurre cuando se dibuja el mismo píxel opaco más de una vez. Por ejemplo, dibujar cartas superpuestas una sobre otra. Esto desperdicia tiempo de renderizado y recursos de hardware, lo que puede resultar en un renderizado lento y tartamudeos, especialmente en dispositivos menos potentes.

Consecuencias del Overdraw

El overdraw afecta negativamente el rendimiento porque:

- Dibuja píxeles invisibles, desperdiciando recursos de hardware.
- Puede resultar en un renderizado lento y tartamudeos.

Cómo Reducir el Overdraw

1. **Eliminar Fondos Innecesarios**

   Elimina fondos que estén cubiertos o invisibles.

   ```
   <ImageView
     android:layout_width="match_parent"
     android:layout_height="match_parent"
     android:src="@drawable/beach"
     android:background="@android:color/white" />
   ```

2. **Recortar Vistas Personalizadas**

   Dibuja solo las porciones visibles de una vista o forma. Sin embargo, el recorte también consume tiempo, por lo que debe hacerse con cuidado.

- **Funciones para Recortar**
  - canvas.clipRect: Define un rectángulo de recorte. Solo se dibuja lo que está dentro del rectángulo.
  - canvas.quickReject: Determina si el contenido a dibujar está dentro de una región recortada. Si lo está, puedes omitirlo y ahorrar tiempo de procesamiento para dibujos complejos.

3. **Reducir la Transparencia**

   El renderizado alfa es el proceso de combinar una imagen con un fondo para crear la apariencia de transparencia parcial o total. Las animaciones con desvanecimientos, sombras y transparencias usan renderizado alfa. Úsalo con cuidado y considera diseños alternativos.

4. **Dibujar la Forma Correcta**

   Considera cómo componer una pantalla para minimizar la superposición. El dibujo correcto es mucho más económico que el recorte.

Simplificación de Jerarquías de Vistas Complejas

Jerarquías de Vistas Complejas

Las jerarquías de vistas complejas son aquellas con muchas vistas anidadas y grupos de vistas. Tienen grandes cantidades de vistas y vistas superpuestas.

Uso de ConstraintLayout

ConstraintLayout ayuda a construir una interfaz de usuario receptiva y aplanar la jerarquía de vistas, optimizándola por el sistema Android.

Eliminar Vistas Innecesarias

Elimina vistas que estén completamente cubiertas, nunca se muestren o estén fuera de la pantalla.

Combinar Vistas

Combinas vistas para usar menos fuentes y estilos y combinar TextViews. Sin embargo, esto puede afectar cómo presentas la información, por lo que debes considerar las compensaciones de diseño y priorizar la simplicidad.

Pases de Diseño Exponenciales

Algunos contenedores de diseño y sus hijos requieren dos pases de diseño para finalizar las posiciones de las vistas secundarias. Cuando los anidas, el número de pases de diseño aumenta exponencialmente con cada nivel de la jerarquía.

- **Cuidado con los Pases de Diseño en**:
  - RelativeLayout
  - LinearLayout que usa measureWithLargestChild
  - GridView que usa gravedad
  - Grupos de vistas personalizadas que son subclases de los anteriores
  - Pesos en LinearLayout (a veces)

Herramientas de Depuración de GPU Overdraw

Inicio de la Herramienta Debug GPU Overdraw

1. En el dispositivo, ve a **Configuración > Opciones de desarrollador**.
2. Desplázate a la sección de **Renderización acelerada por hardware**.
3. Selecciona **Depurar Overdraw de GPU**.
4. Selecciona **Mostrar áreas de overdraw**.

Colores y sus Significados

- **Color verdadero**: Sin overdraw.
- **Púrpura/Azul**: Overdraw una vez.
- **Verde**: Overdraw dos veces.
- **Rosa**: Overdraw tres veces.
- **Rojo**: Overdraw cuatro o más veces.

Herramientas de Inspección de Layout

Uso de la Herramienta Layout Inspector

Usa Layout Inspector para inspeccionar la jerarquía de vistas en tiempo de ejecución, especialmente cuando tu diseño se construye en tiempo de ejecución y no en XML. Crea una instantánea estática de la jerarquía de vistas en un momento dado.

Inicio de la Herramienta Layout Inspector

1. Ejecuta la aplicación.
2. En Android Studio, selecciona **Tools > Android > Layout Inspector**.

3. Elige el proceso de la aplicación al que adjuntar (por ejemplo, com.example.android.stackedview).

Componentes Básicos de Layout Inspector

- **Pantalla del dispositivo**: Muestra la disposición de las vistas.
- **Jerarquía de vistas**: Muestra la estructura de la jerarquía de vistas.
- **Propiedades de la vista seleccionada**: Muestra las propiedades de la vista seleccionada.

Herramienta Lint

Usa la herramienta Lint en archivos de diseño para encontrar optimizaciones en la jerarquía de vistas. Lint se ejecuta automáticamente cada vez que compilas tu programa, pero también puedes ejecutarlo manualmente seleccionando **Analyze > Inspect Code**.

Ejemplo de Mensaje de Inspección de Lint

"El nodo puede ser reemplazado por TextView con drawable compuesto. Un LinearLayout que contiene un ImageView y un TextView puede ser manejado de manera más eficiente como un drawable compuesto. Agrega un Drawable a un TextView usando uno de los métodos setCompoundDrawables y especifica cómo el texto fluye alrededor del drawable."

Herramienta Systrace

Uso de Systrace

Systrace está integrado en el SDK de Android y captura y muestra los tiempos de ejecución de la aplicación y los procesos del sistema Android. Combina datos del kernel de Android como el programador de CPU, la actividad del disco y los hilos de la aplicación para generar un informe HTML.

Inicio de Systrace desde Android Studio

1. Ve a **Tools > Android > Android Device Monitor**.
2. Haz clic en **DDMS (1)**.
3. Elige el paquete (2).
4. Haz clic en **Systrace (3)**.

Salida de Systrace

Abre el archivo trace.html en Chrome para ver los detalles del rendimiento.

Herramienta dumpsys

Uso de dumpsys

El comando dumpsys se ejecuta en el dispositivo y descarga información de estado sobre los servicios del sistema. Puedes obtener información sobre los fotogramas con los siguientes comandos:

adb shell dumpsys gfxinfo <NOMBRE_DEL_PAQUETE>
adb shell dumpsys gfxinfo com.example.android.largeimages
adb shell dumpsys gfxinfo com.example.android.largeimages framestats (solo 6.0+)

5.3 Optimización de Memoria en Android

Fundamentos de la Memoria en Android

Uso de Memoria en Android

Todas las aplicaciones, servicios y procesos requieren memoria para almacenar instrucciones y datos. Una aplicación en ejecución asigna memoria para objetos y procesos en su heap de memoria asignado.

- **Asignación de memoria**: Proceso de reservar memoria para los objetos y procesos de tu aplicación.

Gestión Automática de Memoria

Android maneja el entorno de memoria de manera automática. El sistema realiza un trabajo considerable para hacer la gestión de memoria lo más rápida y eficiente posible.

- **Sistema operativo Android**: Administra la memoria para ti, eliminando la necesidad de manejar la memoria manualmente.

Limitaciones del Sistema de Gestión de Memoria

Aunque Android gestiona la memoria, la eficiencia del código es crucial. Un código ineficiente en términos de memoria puede afectar el rendimiento global de la aplicación. La limpieza de recursos de memoria también consume tiempo, afectando el rendimiento.

Objetos Referenciados en Memoria

Android Runtime (ART) y Dalvik Virtual Machine utilizan paginación y mapeo de memoria. La memoria que una aplicación modifica permanece en la RAM hasta que se liberan las referencias a los objetos, permitiendo su recolección (garbage collection).

Liberación de Referencias

Para hacer que la memoria esté disponible para la recolección de basura, es esencial liberar las referencias a los objetos que ya no son necesarios.

Recolección de Basura (Garbage Collection)

Definición

La recolección de basura (GC) es el proceso automático de liberar espacio en la memoria de una computadora eliminando datos que ya no se necesitan o no se usan. Encuentra objetos de datos que no se pueden acceder en el futuro porque ya no se referencian y reclama los recursos usados por esos objetos.

Impacto en el Rendimiento

Aunque no se puede controlar directamente el proceso de GC, el flujo de código afecta las condiciones que lo desencadenan. La recolección frecuente de basura puede perjudicar el rendimiento de la aplicación, causando ralentizaciones y posibles errores de memoria insuficiente (OutOfMemoryError).

Ejemplo de Asignaciones en Bucles

Asignar y mantener referencias a múltiples objetos dentro de la parte más interna de un bucle puede desencadenar eventos frecuentes de GC, afectando negativamente el rendimiento.

Fugas de Memoria y Churn

Fugas de Memoria

Una fuga de memoria ocurre cuando el código asigna memoria para muchos objetos y nunca libera las referencias, lo que lleva a que menos memoria esté disponible, ralentizando la aplicación y eventualmente causando fallos.

Churn de Memoria

El churn de memoria se refiere a la condición en la que la aplicación está baja en memoria, aumentando la frecuencia de eventos de recolección de basura, lo que a su vez toma más tiempo y ralentiza la aplicación.

Prevención de Fugas de Memoria

No Filtrar Objetos de Vista

Evita referenciar objetos de vista desde fuera del hilo de la interfaz de usuario, en callbacks asíncronos o desde objetos estáticos.

Evitar Asignaciones en Bucles

No asignes objetos dentro de bucles internos. En su lugar, realiza las asignaciones fuera del bucle o rediseña el código para evitar la asignación.

Evitar Asignaciones en onDraw()

El método onDraw() se llama 60 veces por segundo. No crees nuevos objetos dentro de onDraw(). En su lugar, crea un objeto en onCreate() y reutilízalo o usa pools de objetos.

Herramienta Memory Profiler

Funcionalidades del Memory Profiler

- Ver el conteo en tiempo real de objetos asignados y eventos de recolección de basura en la línea de tiempo.
- Volcar el heap de Java para ver qué objetos están utilizando memoria en un momento dado.
- Grabar asignaciones de memoria para ver dónde tu código está asignando demasiados objetos o objetos grandes.

Iniciar el Memory Profiler

1. Asegúrate de que las opciones de desarrollador estén habilitadas en el dispositivo.
2. En Android Studio:
    - Abre el **Android Profiler**.
    - Selecciona el dispositivo y la aplicación.
    - El gráfico de MEMORIA empieza.

Gráficos de Memoria

Gráfico Inicial

Muestra la memoria total utilizada a lo largo del tiempo. Haz clic para expandir y ver detalles.

Gráficos de Memoria Expandidos

Muestra un gráfico apilado para los tipos de memoria y los eventos de GC representados con iconos de contenedores de basura.

Uso de las Herramientas

1. **Forzar recolección de basura**.
2. **Capturar un volcado de heap y mostrar su contenido**.
3. **Grabar asignaciones de memoria y mostrar los datos registrados**.

Grabación de Asignaciones

Muestra los resultados de la grabación de asignaciones y el volcado de heap por debajo de la línea de tiempo, permitiendo seleccionar nombres de clase para ver instancias y detalles en el panel de referencia.

Volcado de Heap de Java

Captura y Análisis del Heap

Haz clic en el botón **Dump Java Heap** en la vista de gráficos detallados. Para inspeccionar el heap volcado, selecciona el nombre de la clase y luego la instancia para ver detalles en el panel de referencias.

Grabación de Asignaciones de Memoria

Procedimiento de Grabación

Haz clic en **Record Memory Allocations** en la vista de gráficos detallados, y luego en **Stop Recording**. El gráfico muestra la porción grabada y la lista de nombres de clase.

Identificación de Asignaciones en el Código

Selecciona una instancia y luego un método en la pila de llamadas para examinar el código correspondiente.

5.4 Rendimiento en Red, Batería y Compresión en Android

Rendimiento en la Red

Consideraciones sobre la Red

Para la mayoría de los usuarios, las redes móviles presentan limitaciones como:

- **Ancho de banda limitado**: Las conexiones móviles a menudo tienen un ancho de banda más bajo en comparación con Wi-Fi.
- **Planes de datos caros y limitados**: Los usuarios a menudo tienen un límite en la cantidad de datos que pueden usar sin incurrir en costos adicionales.
- **Drenaje de la batería**: El uso continuo de la red puede consumir una cantidad significativa de batería.

Mejores Prácticas para la Red

1. **Optimizar para Conexiones de Baja Velocidad o Inconsistentes**
   - Asegúrate de que tu aplicación funcione correctamente en conexiones lentas o cuando no hay conexión.
   - Almacena datos localmente para permitir el trabajo sin conexión.
2. **Redimensionar Imágenes y Optimizar su Calidad**
   - Ajusta el tamaño de las imágenes antes de enviarlas a través de la red.
   - Elige el mejor formato de codificación para equilibrar la calidad y el tamaño del archivo.
3. **Reducir el Tiempo Activo del Radio**
   - Optimiza la frecuencia de las solicitudes de red.
   - Agrupa las solicitudes para minimizar el número de veces que el radio se enciende.
4. **Reducir el Tamaño de los Datos por Solicitud**
   - Envía solo los datos necesarios.
   - Utiliza compresión de datos para reducir el tamaño de las solicitudes y respuestas.

Funcionamiento del Radio Móvil

Dentro de tu dispositivo móvil, hay un pequeño chip de radio cuyo propósito es comunicarse con las torres de celular locales y transmitir datos.

Estados de Energía del Radio Móvil

- **Full Power**: Conexión activa, permitiendo la transferencia de datos a la tasa más alta posible.
- **Low Power**: Utiliza aproximadamente un 50% menos de energía.
- **Standby**: Estado de energía mínima cuando no se requiere una conexión de red activa.

El estado de energía del radio varía según el operador y el hardware del dispositivo.

Latencia vs. Uso de Energía

- **Low Power y Standby**: Drenan mucha menos batería pero introducen una latencia significativa.
- **Full Power desde Low Power**: Toma aproximadamente 1.5 segundos.
- **Full Power desde Standby**: Puede tomar más de 2 segundos.

Minimizar Encendidos del Radio

- **Minimizar el Número de Encendidos**: Envía la mayor cantidad de datos posible por ciclo y obtén respuestas del servidor antes de que el radio se apague.
- **API Disponibles**: JobScheduler, Firebase JobDispatcher.

Optimización de Imágenes

Mejores Prácticas para la Optimización de Imágenes

1. **Optimización de Imágenes PNG**
   - Reduce el número de colores únicos.
   - Ajusta la calidad para equilibrar el tamaño del archivo y la calidad visual.
2. **Optimización de Imágenes JPG**
   - Ajusta la calidad alrededor del 75% para un tamaño de imagen significativamente menor sin una diferencia visual significativa.
3. **Uso del Formato WebP**
   - WebP es un formato de imagen de Google que proporciona compresión con pérdida (como JPEG) y transparencia (como PNG) pero con mejor compresión que ambos.

Selección del Formato de Imagen

- **WebP**: Si es compatible.
- **PNG**: Si la imagen necesita transparencia.
- **JPG**: Si la imagen no necesita transparencia y es compleja en términos de colores y estructura.

Prácticas para Cargar Imágenes

- **Equilibrar Tamaño y Calidad**: Ajusta la resolución y calidad de la imagen según sea necesario.
- **Backend de Servicio de Imágenes**: Utiliza servicios como Google App Engine para manejar imágenes.
- **Librerías de Carga de Imágenes**: Usa librerías como Glide y Picasso para simplificar y optimizar la carga de imágenes.

Conversión de Imágenes a WebP

Android ha incluido soporte para WebP con pérdida desde Android 4.0 (API 14) y soporte para WebP sin pérdida y transparente desde Android 4.2 (API 18).

Conversión en Android Studio

1. En res/drawable, haz clic derecho en la imagen.
2. Elige Convert to WebP.
3. Ajusta los parámetros en el diálogo de conversión.
4. Haz clic en OK para previsualizar.
5. Ajusta la calidad y tamaño.
6. Haz clic en Finish.

Compresión de Datos de Texto

Lista de Verificación

1. **Editar**: Reducir el tamaño de la página, menos palabras, solo contenido relevante, múltiples páginas más pequeñas.
2. **Minificar**: Usar minificadores de CSS y JavaScript.
3. **Comprimir**: Habilitar la compresión GZIP en el servidor.
4. **Compresión Offline**: Usar Zopfli o 7-Zip para la compresión offline.

Serialización de Datos

Serialización

Los datos estructurados deben convertirse a un formato que pueda almacenarse o enviarse a través de la red y utilizarse para reconstruir los datos originales y sus estados en el destino. Ejemplos comunes son JSON y XML, que son legibles por humanos pero voluminosos y lentos.

FlatBuffers

FlatBuffers permite crear un esquema que describe tus datos y compilarlo en código fuente que puede serializar y deserializar tus datos de manera más eficiente.

Herramientas de Análisis

Network Profiler

Network Profiler es parte de Android Profiler y permite:

- Rastrear en tiempo real cuándo tu aplicación hace solicitudes de red.
- Monitorear cómo y cuándo tu aplicación transfiere datos.
- Reducir la frecuencia de las transferencias de datos.
- Optimizar los datos transferidos durante cada conexión.

Iniciar Network Profiler

1. Abre Android Profiler en Android Studio.
2. Visualiza el gráfico de NETWORK.
3. Haz clic en el gráfico para ver los detalles.

Optimización del Uso de la Batería

batterystats y Battery Historian

dumpsys batterystats recoge eventos relacionados con el consumo de energía desde los registros del sistema del dispositivo. Battery Historian crea una visualización que puedes ver en el navegador Chrome.

Obtener Datos de la Batería

1. Restablecer el registro de la batería: adb shell dumpsys batterystats --reset.
2. Volcar datos de la batería y guardarlos en un archivo de texto: adb shell dumpsys batterystats > batterystats.txt.
3. Crear un informe: adb bugreport bugreport.zip (7.0+) o adb bugreport > bugreport.txt (6.0-).

Abrir el Informe en el Navegador

1. Inicia Docker para ejecutar Battery Historian.
2. Abre una pestaña en tu navegador.

3. Sube el archivo bugreport.zip o bugreport.txt.

6. Conclusiones

En esta unidad didáctica, hemos profundizado en la optimización del rendimiento de las aplicaciones Android, un aspecto fundamental para mejorar la experiencia del usuario y asegurar la eficiencia del software. A través de las diversas técnicas presentadas, comprendimos cómo gestionar mejor los recursos del dispositivo, desde la memoria hasta el uso de la CPU, la batería y la red.

El uso de **herramientas de perfilado** como Android Profiler nos permite identificar y corregir cuellos de botella en el rendimiento. Estas herramientas son esenciales para monitorear el uso de memoria, el rendimiento de la CPU, el consumo de batería y la transmisión de datos en tiempo real, proporcionando a los desarrolladores una visión clara de las áreas que requieren optimización.

Asimismo, aprendimos la importancia de **optimizar el uso de memoria**, lo que implica reducir las fugas de memoria y el número de eventos de recolección de basura, técnicas que aseguran que las aplicaciones no se ralenticen ni se detengan debido a una mala gestión de los recursos.

Otra parte fundamental de esta unidad es la **optimización del uso de la red y la batería**, donde se resaltó la importancia de agrupar solicitudes de red y utilizar la compresión de datos para minimizar el consumo de energía y datos móviles.

Finalmente, con el análisis del **uso de GPU y el manejo de gráficos**, se destacó la relevancia de evitar el overdraw y de reducir la complejidad de las jerarquías de vistas, lo que mejora la fluidez y la rapidez de las interfaces de usuario.

En conclusión, la optimización de aplicaciones no solo mejora la experiencia del usuario, sino que también asegura que las aplicaciones funcionen de manera eficiente en dispositivos de diferentes capacidades. Estos conceptos son clave para desarrollar software móvil competitivo y robusto.

7. Bibliografía

Android Developers. (2023). *Optimize for Performance*. Google. Disponible en: https://developer.android.com/topic/performance

Phillips, B., Stewart, C., Hardy, K., & Marsicano, B. (2019). *Android Programming: The Big Nerd Ranch Guide*. Big Nerd Ranch.

Meier, R. (2015). *Professional Android: Building Apps with Android Studio*. John Wiley & Sons.

Goyal, A. (2018). *Efficient Android Threading: Asynchronous Processing Techniques for Android Applications*. Apress.

Ferrill, P. (2017). *Optimizing Android Apps: Improve Performance with the Android Profiler*. Packt Publishing.

Estas fuentes proporcionan las mejores prácticas y estrategias clave para optimizar el rendimiento de las aplicaciones Android, lo que contribuye a una experiencia de usuario más fluida y satisfactoria.

UAX Universidad Alfonso X el Sabio

G R A C I A S

UAX.COM