Desarrollo de aplicaciones móviles I

Tema Nº11:Conexión a Servicios Web con Retrofit en Android.

Indicador de logro Nº11:Consume servicios web desde una aplicación en Android mediante el uso de la librería Retrofit.

**TEMA 01 Teoría de los**

Imagen que contiene Icono

Descripción generada automáticamente

**TEMA Nº11:**

Conexión a Servicios Web con Retrofit en Android.

**Subtema 11.1:**

¿Cómo funciona un Servicio Web en Android?

**Ejemplos:**

Un web service facilita un servicio a través de Internet: se trata de una interfaz mediante la que dos máquinas (o aplicaciones) se comunican entre sí. Esta tecnología se caracteriza por estos dos rasgos:

* Multiplataforma: cliente y servidor no tienen por qué contar con la misma configuración para comunicarse. El servicio web se encarga de hacerlo posible.
* Distribuida: por lo general, un servicio web no está disponible para un único cliente, sino que son diferentes los que acceden a él a través de Internet.

Cuando se utiliza un web service, un cliente manda una solicitud a un servidor, desencadenando una acción por parte de este. A continuación, el servidor devuelve una respuesta al cliente.

**La tecnología detrás de un servicio web con ejemplo**

Todos los web services cuentan con un Uniform Resource Identifier (URI) unívoco, esto es, la dirección del servicio web. Es similar al Uniform Resource Locator (URL) que permite acceder a páginas web. El catálogo UDDI debía desempeñar también un papel importante, pues permitía encontrar los servicios web, pero este servicio nunca logró imponerse y sus mayores partidarios terminaron retirándose del proyecto.

Más importancia tiene el lenguaje Web Service Description Language (WSDL). Un servicio web contiene un archivo en WSDL en el que se describe el servicio de forma detallada. Con esta información, el cliente puede comprender qué funciones puede ejecutar en el servidor a través del servicio web. La comunicación funciona exclusivamente mediante diferentes protocolos y arquitecturas. Entre ellos, son muy populares el protocolo de red SOAP en combinación con el estándar de Internet HTTP o los servicios web basados en una arquitectura REST.

Con estas tecnologías se posibilita el intercambio de peticiones y respuestas a menudo utilizando el lenguaje de marcado extensible (XML). Este lenguaje, relativamente simple, puede ser interpretado en igual medida por personas y ordenadores, y además es adecuado para unir sistemas con requisitos diferentes. Con todo, REST también admite otros formatos, como JSON.

Veamos cómo funciona la mecánica de esta tecnología con un ejemplo de web service. Partamos de un software escrito en Visual Basic que se ejecuta en una máquina con sistema Windows. El programa necesita el servicio de un servidor web Apache. Para ello, el cliente envía una solicitud SOAP en forma de mensaje HTTP al servidor. El web service interpreta el contenido de la solicitud y se encarga de que el servidor lleve a cabo una acción. Finalmente, el servicio web formula una respuesta y la envía de vuelta al cliente (de nuevo con SOAP y HTTP), que vuelve a interpretarla. La información se envía entonces al software, donde será procesada.

**Ventajas y desventajas de los servicios web**

La ventaja principal de los servicios web es que la comunicación no depende de una plataforma determinada, por lo que el cliente y el servidor apenas han de presentar rasgos en común para poder comunicarse. Para ello, la tecnología web service recurre a formatos estandarizados que interpretan todos los sistemas.

Pero en estos formatos es donde encontramos una de las desventajas. Precisamente, XML es un formato más bien voluminoso que genera grandes paquetes de datos, lo que puede crear problemas en las conexiones de red lentas. Otra posibilidad que permite conectar a dos sistemas a través de Internet son las API web. Aunque, por lo general, son más rápidas, someten a cliente y servidor a especificaciones más concretas, con lo que la interoperabilidad se ve limitada.

**Subtema 11.2:**

Qué es un API Rest, Verbos Http y Endpoints

**Ejemplos:**

REST cambió por completo la ingeniería de software a partir del 2000. Este nuevo enfoque de desarrollo de proyectos y servicios web fue definido por Roy Fielding, el padre de la especificación HTTP y uno los referentes internacionales en todo lo relacionado con la Arquitectura de Redes, en su disertación ‘Architectural Styles and the Design of Network-based Software Architectures’. En el campo de las APIs, REST (Representational State Transfer- Transferencia de Estado Representacional) es, a día de hoy, el alfa y omega del desarrollo de servicios de aplicaciones.

En la actualidad no existe proyecto o aplicación que no disponga de una API REST para la creación de servicios profesionales a partir de ese software. Twitter, YouTube, los sistemas de identificación con Facebook… hay cientos de empresas que generan negocio gracias a REST y las APIs REST. Sin ellas, todo el crecimiento en horizontal sería prácticamente imposible. Esto es así porque REST es el estándar más lógico, eficiente y habitual en la creación de APIs para servicios de Internet.

Buscando una definición sencilla, REST es cualquier interfaz entre sistemas que use HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON. Es una alternativa en auge a otros protocolos estándar de intercambio de datos como SOAP (Simple Object Access Protocol), que disponen de una gran capacidad pero también mucha complejidad. A veces es preferible una solución más sencilla de manipulación de datos como REST.

Características de REST

* Protocolo cliente/servidor sin estado: cada petición HTTP contiene toda la información necesaria para ejecutarla, lo que permite que ni cliente ni servidor necesiten recordar ningún estado previo para satisfacerla. Aunque esto es así, algunas aplicaciones HTTP incorporan memoria caché. Se configura lo que se conoce como protocolo cliente-caché-servidor sin estado: existe la posibilidad de definir algunas respuestas a peticiones HTTP concretas como cacheables, con el objetivo de que el cliente pueda ejecutar en un futuro la misma respuesta para peticiones idénticas. De todas formas, que exista la posibilidad no significa que sea lo más recomendable.
* Las operaciones más importantes relacionadas con los datos en cualquier sistema REST y la especificación HTTP son cuatro: POST (crear), GET (leer y consultar), PUT (editar) y DELETE (eliminar).
* Los objetos en REST siempre se manipulan a partir de la URI. Es la URI y ningún otro elemento el identificador único de cada recurso de ese sistema REST. La URI nos facilita acceder a la información para su modificación o borrado, o, por ejemplo, para compartir su ubicación exacta con terceros.
* Interfaz uniforme: para la transferencia de datos en un sistema REST, este aplica acciones concretas (POST, GET, PUT y DELETE) sobre los recursos, siempre y cuando estén identificados con una URI. Esto facilita la existencia de una interfaz uniforme que sistematiza el proceso con la información.
* Sistema de capas: arquitectura jerárquica entre los componentes. Cada una de estas capas lleva a cabo una funcionalidad dentro del sistema REST.
* Uso de hipermedios: hipermedia es un término acuñado por Ted Nelson en 1965 y que es una extensión del concepto de hipertexto. Ese concepto llevado al desarrollo de páginas web es lo que permite que el usuario puede navegar por el conjunto de objetos a través de enlaces HTML. En el caso de una API REST, el concepto de hipermedia explica la capacidad de una interfaz de desarrollo de aplicaciones de proporcionar al cliente y al usuario los enlaces adecuados para ejecutar acciones concretas sobre los datos.

Para cualquier API REST es obligatorio disponer del principio HATEOAS (Hypermedia As The Engine Of Application State – Hipermedia Como Motor del Estado de la Aplicación) para ser una verdadera API REST. Este principio es el que define que cada vez que se hace una petición al servidor y éste devuelve una respuesta, parte de la información que contendrá serán los hipervínculos de navegación asociada a otros recursos del cliente.

Ventajas que ofrece REST para el desarrollo

1. Separación entre el cliente y el servidor: el protocolo REST separa totalmente la interfaz de usuario del servidor y el almacenamiento de datos. Eso tiene algunas ventajas cuando se hacen desarrollos. Por ejemplo, mejora la portabilidad de la interfaz a otro tipo de plataformas, aumenta la escalabilidad de los proyectos y permite que los distintos componentes de los desarrollos se puedan evolucionar de forma independiente.
2. Visibilidad, fiabilidad y escalabilidad. La separación entre cliente y servidor tiene una ventaja evidente y es que cualquier equipo de desarrollo puede escalar el producto sin excesivos problemas. Se puede migrar a otros servidores o realizar todo tipo de cambios en la base de datos, siempre y cuando los datos de cada una de las peticiones se envíen de forma correcta. Esta separación facilita tener en servidores distintos el front y el back y eso convierte a las aplicaciones en productos más flexibles a la hora de trabajar.
3. La API REST siempre es independiente del tipo de plataformas o lenguajes: la API REST siempre se adapta al tipo de sintaxis o plataformas con las que se estén trabajando, lo que ofrece una gran libertad a la hora de cambiar o probar nuevos entornos dentro del desarrollo. Con una API REST se pueden tener servidores PHP, Java, Python o Node.js. Lo único que es indispensable es que las respuestas a las peticiones se hagan siempre en el lenguaje de intercambio de información usado, normalmente XML o JSON.

Retrofit:

* Retrofit es un cliente REST para Android y Java que permite realizar peticiones a servicios web de manera sencilla
* Se encarga de procesar la petición y realizar las transformaciones necesarias tanto el cliente como el servidor de tal manera que ambos hablen el mismo idioma



* Con Retrofit podemos representar una petición a una API dentro de una interfaz de Java
* Retrofit hace uso de anotaciones para indicar rutas, métodos, cabeceras y contenido de cada petición
* Para realizar una petición primero debemos crear una implementación de la interfaz, finalmente creamos la petición y esperamos la promesa de respuesta.

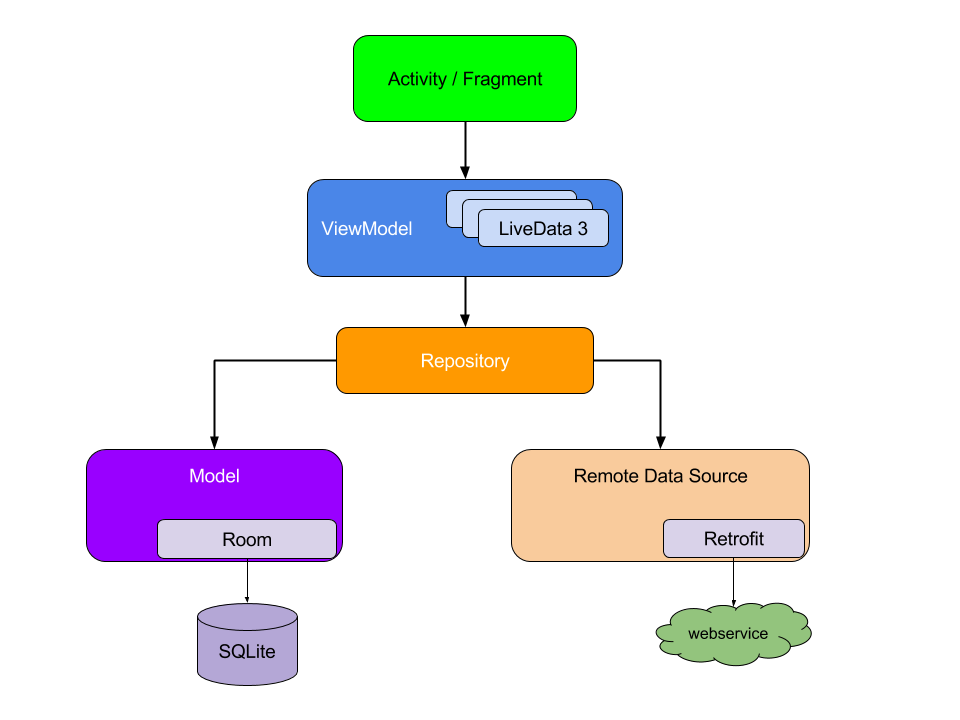
Texto

Descripción generada automáticamente con confianza baja

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Retrofit dentro de la arquitectura de componentes de Android



Peticiones web con Retrofit

Diagrama

Descripción generada automáticamente

Configuración de Retrofit:

Texto

Descripción generada automáticamente

Retrofit: DAO.

Texto

Descripción generada automáticamente

Petición desde la Actividad.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**Actividad:**

1. CUESTIONARIO TÉCNICO

* ¿Qué es una API Rest?
* ¿Para que sirve retrofit?
* Desarrollar una aplicación de Android que al lanzar la actividad de inicio muestre una lista de recursos
* La aplicación debe comunicarse con un servicio web utilizando la librería Retrofit
* Utilizar la estructura de proyecto definida en los componentes de arquitectura de Android

1. CONCLUSIONES DE LA EXPERIENCIA

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_