Contenido

Introducción

Los aplicaciones móviles han tomado una gran relevancia en el área de desarrollo lo que antes parecía un futuro lejano a tomado un rumbo muy acelerado de las tecnologías y la mayoría de las aplicaciones deben contener una gran cantidad de datos guardados, tienen que persistir datos en algún momento, ya sea serializándolos, guardándolos en una base de datos relacional, o una base de datos orientada a objetos, etc. Para hacer esto, la aplicación interactúa con la base de datos. El "como interactúa" NO debe ser asunto de la capa de lógica de negocio de la aplicación, ya que para eso está la capa de persistencia, que es la encargada de interactuar con la base de datos.

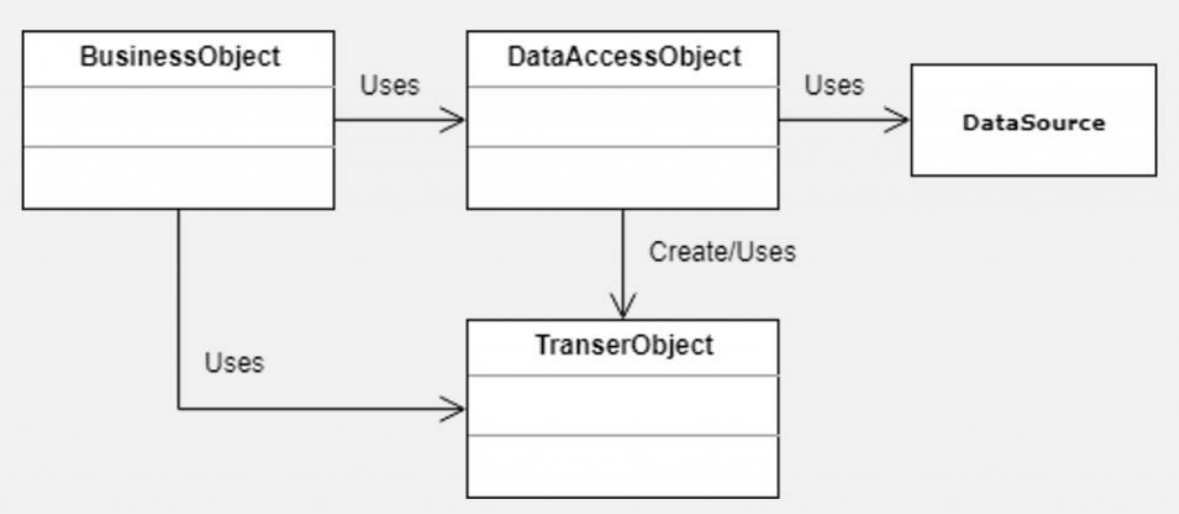
En la presente documentación daremos a conocer todo lo que se encuentra detrás de la lógica de la conexión a la base de datos y los conceptos claves de una buena comunicación con el dispositivo móvil.

Modelado de objetos de acceso a datos en dispositivos móviles

Un Objeto de Acceso a Datos o Data Access Object (DAO) son una serie de objetos que le permiten tener acceso y manipular datos mediante programación en bases de datos locales o remotos. Puede utilizar DAO para administrar bases de datos, así como sus objetos y su estructura. Es un componente de software que suministra una interfaz común entre la aplicación y uno o más dispositivos de almacenamiento de datos, tales como una Base de datos o un archivo. El término se aplica frecuentemente al Patrón de diseño Object.Características de los objetos de acceso a datos en dispositivos móviles.

Proceso de modelado de objetos de acceso a datos en dispositivos móviles.

Dado lo anterior**,** el patrón DAO propone separar por completo la lógica de negocio de la lógica para acceder a los datos, de esta forma, el DAO proporcionará los métodos necesarios para insertar, actualizar, borrar y consultar la información; por otra parte, la capa de negocio solo se preocupa por lógica de negocio y utiliza el DAO para interactuar con la fuente de datos.

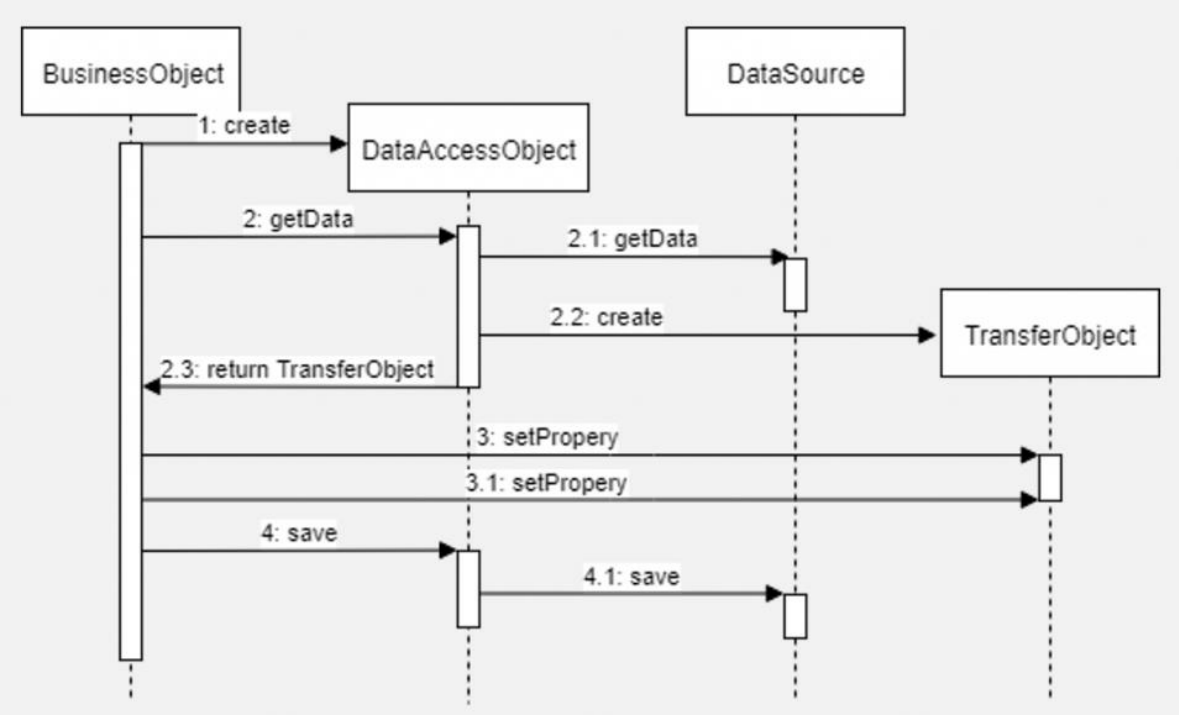


Los compones que conforman el patrón son:

- **BusinessObject**: representa un objeto con la lógica de negocio.

- **DataAccessObject**: representa una capa de acceso a datos que oculta la fuente y los detalles técnicos para recuperar los datos.
- **TransferObject**: este es un objeto plano que implementa el patrón Data Transfer Object (DTO), el cual sirve para transmitir la información entre el DAO y el Business Service.
- **DataSource**: representa de forma abstracta la fuente de datos, la cual puede ser una base de datos, Webservices, LDAP, archivos de texto, etc.

El siguiente diagrama muestra mejor la forma en que funciona el patrón, pues muestra de forma secuencial la forma en que se ejecutaría el patrón.



Proceso de programación de objetos de acceso a datos en dispositivos móviles.

Utilizando Room

Configuraremos primero para usar Room en tu app, agrega las siguientes dependencias al archivo build.gradle de la app:

Groovy Kotlin

```
dependencies {
    def room_version = "2.4.2"

    implementation "androidx.room:room-runtime:$room_version"
    annotationProcessor "androidx.room:room-compiler:$room_version"

    // optional - RxJava2 support for Room
    implementation "androidx.room:room-rxjava2:$room_version"

    // optional - RxJava3 support for Room
    implementation "androidx.room:room-rxjava3:$room_version"

    // optional - Guava support for Room, including Optional and ListenableFuture
    implementation "androidx.room:room-guava:$room_version"

    // optional - Test helpers
    testImplementation "androidx.room:room-testing:$room_version"

    // optional - Paging 3 Integration
    implementation "androidx.room:room-paging:2.5.0-alpha02"
}
```

Componentes principales

Estos son los tres componentes principales de Room:

- La clase de la base de datos que contiene la base de datos y sirve como punto de acceso principal para la conexión subyacente a los datos persistentes de la app.
- Las entidades de datos que representan tablas de la base de datos de tu app.
- Objetos de acceso a datos (DAOs) que proporcionan métodos que tu app puede usar para consultar, actualizar, insertar y borrar datos en la base de datos.

Entidad de datos

El siguiente código define una entidad de datos User. Cada instancia de User representa una fila en una tabla de user en la base de datos de la app.

```java
@Entity
public class User {
    @PrimaryKey
    public int uid;

    @ColumnInfo(name = "first_name")
    public String firstName;

    @ColumnInfo(name = "last_name")
    public String lastName;
}
```

Objeto de acceso a datos (DAO)

El siguiente código define un DAO llamado UserDao. UserDao proporciona los métodos que el resto de la app usa para interactuar con los datos de la tabla user.

```java
@Dao
public interface UserDao {
    @Query("SELECT * FROM user")
    List<User> getAll();

    @Query("SELECT * FROM user WHERE uid IN (:userIds)")
    List<User> loadAllByIds(int[] userIds);

    @Query("SELECT * FROM user WHERE first_name LIKE :first AND " +
           "last_name LIKE :last LIMIT 1")
    User findByName(String first, String last);

    @Insert
    void insertAll(User... users);

    @Delete
    void delete(User user);
}
```

Database

Con el siguiente código, se define una clase AppDatabase para contener la base de datos. AppDatabase define la configuración de la base de datos y sirve como el

punto de acceso principal de la app a los datos persistentes. La clase de la base de datos debe cumplir con las siguientes condiciones:

- La clase debe tener una anotación @Database que incluya un array entities que enumere todas las entidades de datos asociados con la base de datos.
- Debe ser una clase abstracta que extienda RoomDatabase.
- Para cada clase DAO que se asoció con la base de datos, esta base de datos debe definir un método abstracto que tenga cero argumentos y muestre una instancia de la clase DAO.

```
@Database(entities = {User.class}, version = 1)
public abstract class AppDatabase extends RoomDatabase {
    public abstract UserDao userDao();
}
```

Manipulación de datos en dispositivos móviles

Reconocer el concepto de conexión a bases de datos.

Una conexión a base de datos es un archivo de configuración donde se especifica los detalles de una base de datos como por ejemplo el servidor de base de datos y el usuario, y los parámetros que permiten una conexión

Describir la conexión a bases de datos estáticos, dinámicos, web y locales en dispositivos móviles.

ESTÁTICOS

Una base de datos estática o también conocidos como almacén de datos es aquella cuya función principal es el almacenamiento y registro de datos fijos. Es decir, guarda información que no se va a modificar ni editar con el tiempo.

Se trata de un tipo de bases de datos de solo lectura. Su implementación se suele realizar con el objetivo de registrar datos históricos para poder comparar su evolución a lo largo del tiempo.

DINÁMICOS

Una base de datos dinámica es aquella en la que se almacenan datos que pueden variar con el paso del tiempo. Para adecuarse a estos datos cambiantes, las bases dinámicas permiten realizar operaciones de edición, actualización o borrado de información.

WEB

Una base de datos web es un sistema para almacenar información al que luego se puede acceder a través de un sitio web. Por ejemplo, una comunidad en línea puede tener una base de datos que almacena el nombre de usuario, la contraseña y otros detalles de todos sus miembros.

En su nivel más simple, una base de datos web es un conjunto de una o más tablas que contienen datos. Cada tabla tiene diferentes campos para almacenar información de varios tipos.

LOCALES

Una base de datos en local utiliza una red local (LAN), de manera que la infraestructura y la gestión de dicha base de datos se realiza en la propia organización. Solo pueden acceder a la información los equipos que estén conectados a la red local.

Explicar el proceso de programación de conexión a bases de datos estáticos en dispositivos móviles.

```
[...]

//Se referencian las Clases necesarias para la conexión con el Servidor MySQL
    import java.sql.Connection;
    import java.sql.DriverManager;
    import java.sql.SQLException;

[...]

//Declaramos los componentes necesarios para introducir los datos de conexión al servidor
    private EditText edServidor;
    private EditText edPuerto;
    private EditText edUsuario;
    private EditText edPassword;
    private String baseDatos = "Tienda";

[...]

//Enlazamos los componentes con los recursos definidos en el layout
        edServidor = (EditText)findViewById(R.id.edServidor);
        edPuerto  = (EditText)findViewById(R.id.edPuerto);
        edUsuario = (EditText)findViewById(R.id.edUsuario);
        edPassword = (EditText)findViewById(R.id.edPassword);

[...]
```

```
//Función que establecerá la conexión con el Servidor si los datos introducidos son correcto.
        //Devuelve un valor de verdadero o falso que indicará si se ha establecido la conexión
        public boolean conectarMySQL()
        {
    try{
            //Cargamos el driver del conector JDBC
            Class.forName(driver).newInstance ();
            //Establecemos la conexión con el Servidor MySQL indicándole como parámetros la url construida,
            //la Base de Datos a la que vamos a conectarnos, y el usuario y contraseña de acceso al servidor
            conexionMySQL = DriverManager.getConnection(urlMySQL + baseDatos, user, password);
            //Comprobamos que la conexión se ha establecido
            if(!conexionMySQL.isClosed())
            {
                estadoConexion = true;
                Toast.makeText(MainActivity.this,"Conexión Establecida", Toast.LENGTH_LONG).show();
            }
            }catch(Exception ex)
            {
                Toast.makeText(MainActivity.this,"Error al comprobar las credenciales:" + ex.getMessage(), Toast.LENGTH_LONG).show();
            }
[...]

//Evento On Click que realiza la llamada a la función conectarMySQL() obteniendo el valor de verdadero
        //o falso para la petición de conexión
    public void abrirConexion(View view)
    {
        Intent intent = new Intent(this,ConsultasSQL.class);
        //Si el valor devuelto por la función es true, pasaremos los datos de la conexión a la siguiente Activity
        if(conectarMySQL() == true)
            {
```

```
57                  Toast.makeText(this, "Los datos de conexión introducidos son correctos.", Toas
    t.LENGTH_LONG).show();
58                  intent.putExtra("servidor", edServidor.getText().toString());
59                  intent.putExtra("puerto", edPuerto.getText().toString());
60                  intent.putExtra("usuario", edUsuario.getText().toString());
61                  intent.putExtra("password", edPassword.getText().toString());
62                  intent.putExtra("datos", baseDatos);
63                  startActivity(intent);
64              }
65      }
66
67  [...]
```

En este caso solamente utilizaremos el Select ya que solo podemos leer nuestra información en la base de datos.

```
//Cargamos el driver del conector JDBC
Class.forName(driver).newInstance ();
//Establecemos la conexión con el Servidor MySQL indicándole como parámetros la url construida,
//la Base de Datos a la que vamos a conectarnos, y el usuario y contraseña de acceso al servidor
con = DriverManager.getConnection(urlMySQL + baseDatos, txtUsuario.getText().toString(), "110904");
st = con.createStatement();
//Se ejecutará la consulta indicada en el campo edConsulta
rs = st.executeQuery("Select * from Cliete");
```

Explicar el proceso de programación de conexión a bases de datos dinámicos en dispositivos móviles.

Utilizaremos la conexión del código anterior solamente configuramos a nuestra nueva base de datos.

```
1   //Cargamos el driver del conector JDBC
2   Class.forName(driver).newInstance ();
3   //Establecemos la conexión con el Servidor MySQL indicándole como parámetros la url construida,
4   //la Base de Datos a la que vamos a conectarnos, y el usuario y contraseña de acceso al servidor
5   con = DriverManager.getConnection(urlMySQL + baseDatos, txtUsuario.getText().toString(), "110904");
6   st = con.createStatement();
7   //Se ejecutará la consulta indicada en el campo edConsulta
8   rs = st.executeQuery("INSERT INTO usuario(id,nombre,paterno) VALUES (1,Neri,Alvarez)");
```

Explicar el proceso de programación de conexión a bases de datos locales en dispositivos móviles.

Utilizando ROOM

Configuración

Para usar Room en tu app, agrega las siguientes dependencias al archivo build.gradle de la app:

```
dependencies {
    val roomVersion = "2.4.2"

    implementation("androidx.room:room-runtime:$roomVersion")
    annotationProcessor("androidx.room:room-compiler:$roomVersion")

    // To use Kotlin annotation processing tool (kapt)
    kapt("androidx.room:room-compiler:$roomVersion")
    // To use Kotlin Symbolic Processing (KSP)
    ksp("androidx.room:room-compiler:$roomVersion")

    // optional - Kotlin Extensions and Coroutines support for Room
    implementation("androidx.room:room-ktx:$roomVersion")

    // optional - RxJava2 support for Room
    implementation("androidx.room:room-rxjava2:$roomVersion")

    // optional - RxJava3 support for Room
    implementation("androidx.room:room-rxjava3:$roomVersion")

    // optional - Guava support for Room, including Optional and ListenableFuture
    implementation("androidx.room:room-guava:$roomVersion")

    // optional - Test helpers
    testImplementation("androidx.room:room-testing:$roomVersion")

    // optional - Paging 3 Integration
    implementation("androidx.room:room-paging:2.5.0-alpha02")
}
```

Entidad de datos

El siguiente código define una entidad de datos User. Cada instancia de User representa una fila en una tabla de user en la base de datos de la app.

```
@Entity
public class User {
    @PrimaryKey
    public int uid;

    @ColumnInfo(name = "first_name")
    public String firstName;

    @ColumnInfo(name = "last_name")
    public String lastName;
}
```

Objeto de acceso a datos (DAO)

El siguiente código define un DAO llamado UserDao. UserDao proporciona los métodos que el resto de la app usa para interactuar con los datos de la tabla user.

```
@Dao
public interface UserDao {
    @Query("SELECT * FROM user")
    List<User> getAll();

    @Query("SELECT * FROM user WHERE uid IN (:userIds)")
    List<User> loadAllByIds(int[] userIds);

    @Query("SELECT * FROM user WHERE first_name LIKE :first AND " +
           "last_name LIKE :last LIMIT 1")
    User findByName(String first, String last);

    @Insert
    void insertAll(User... users);

    @Delete
    void delete(User user);
}
```

Database

Con el siguiente código, se define una clase AppDatabase para contener la base de datos. AppDatabase define la configuración de la base de datos y sirve como el punto de acceso principal de la app a los datos persistentes.

```
@Database(entities = {User.class}, version = 1)
public abstract class AppDatabase extends RoomDatabase {
    public abstract UserDao userDao();
}
```

Después de definir la entidad de datos, el DAO y el objeto de base de datos, puedes usar el siguiente código para crear una instancia de la base de datos:

```
AppDatabase db = Room.databaseBuilder(getApplicationContext(),
        AppDatabase.class, "database-name").build();
```

Luego, puedes usar los métodos abstractos de AppDatabase para obtener una instancia del DAO. A su vez, puedes usar los métodos de la instancia del DAO para interactuar con la base de datos:

```
UserDao userDao = db.userDao();
List<User> users = userDao.getAll();
```

Persistencia de datos en los dispositivos móviles.

Concepto de persistencia en dispositivos móviles.

La persistencia en toda la aplicación ya sea una aplicación de Android o cualquier otro tipo de aplicación, consiste en que los datos que maneja la aplicación "sobreviven" a su ejecución en el tiempo; en otras palabras, incluye el almacenamiento de datos en un medio volátil no volátil secundario para posterior reconstrucción y uso, por lo que son temporalmente independientes del proceso que los creó. En términos sencillos... los datos no se eliminan después de cerrar la aplicación.

Retos de la persistencia en los dispositivos móviles.

Las plataformas de desarrollo móvil ofrecen diferentes alternativas a la persistencia de datos, permitiéndole almacenar todo, desde preferencias de aplicaciones hasta documentos o datos complejos. No solo necesitamos almacenar los datos en el dispositivo móvil, en algunos casos también nos interesa guardarlos en el servidor, por ejemplo a través de un servicio REST.

1. Persistencia en Android: Preferencias Compartidas o Shared Preferences

Usando Shared Preferences, podemos almacenar y recuperar información como texto, booleanos y números en un formato clave-valor; esto le permite almacenar configuraciones de usuario como: estilos, preferencias, etc.

2. Persistencia en Android: Almacenar archivos en memoria

Este tipo de persistencia es uno de los más conocidos porque la mayoría de los lenguajes de programación excepto Java los soportan; incluyendo guardar y recuperar información en archivos; Android permite escribir y leer archivos ubicados en la memoria interna del dispositivo; como Preferencias Compartidas.

3. Persistencia en Android: Base de Datos SQLite

SQLite es un sistema de gestión de bases de datos relacionales contenido en una biblioteca relativamente pequeña escrita en C. A diferencia de los sistemas de administración de bases de datos cliente-servidor, el motor SQLite no es un proceso separado que se comunica con el programa principal. En cambio, la biblioteca SQLite está vinculada con el programa y se convierte en una parte integral del mismo.

Algunas características interesantes de SQLite

Toda la base de datos (definiciones, tablas, índices y los propios datos) se guarda como un único archivo estándar en el host.

SQLite usa un sistema de tipos inusual. A diferencia de los tipos que se asignan a columnas en la mayoría de los sistemas de bases de datos SQL, los tipos se asignan a valores individuales. Por ejemplo, una cadena de texto se puede insertar en una columna de tipo entero (aunque SQLite intentará convertir la cadena en un número entero primero).

Formas de persistencia en los sistemas operativos de los dispositivos móviles: preferencias, almacenamiento de archivos, datos estructurados.

Se requiere una API básica para almacenar estructuras de datos. Si la cantidad de información que necesitamos almacenar no es demasiado grande y no necesitamos hacer consultas o búsquedas, podemos usar API de iOS bastante simples (al menos en comparación con las que se usan para acceder a bases de datos).

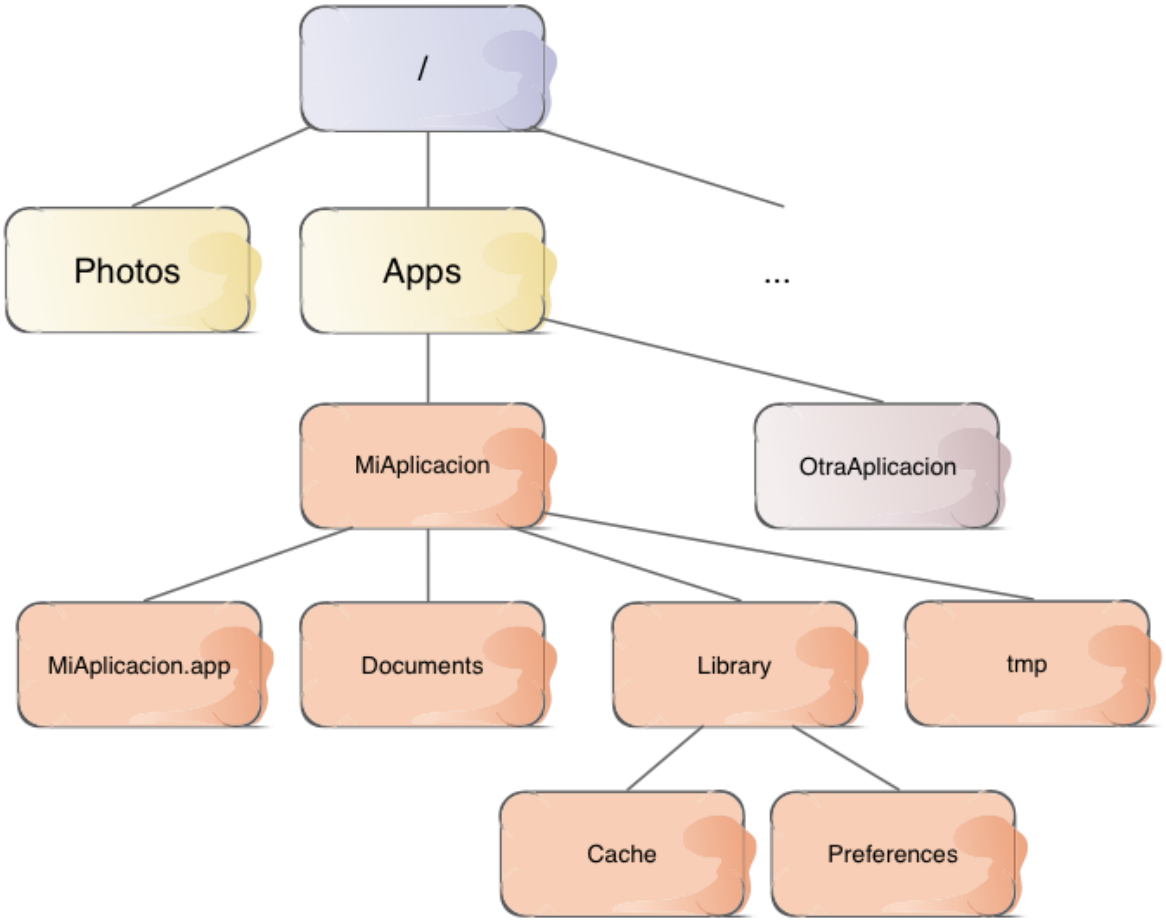
El caso más típico es cuando necesitamos almacenar un conjunto de pares "clave/valor", podemos usar una lista de propiedades o el sistema de preferencias de iOS. Este es un mecanismo común para guardar las preferencias de la aplicación (colores, fuentes, datos básicos del usuario, etc.).

Sistema de archivos

En iOS, el sistema de archivos completo no es visible para ninguna aplicación porque está contenido en un llamado sandbox por razones de seguridad. Una aplicación solo puede acceder a archivos y directorios en su sandbox y, a la inversa, otras aplicaciones no pueden acceder a ellos.

Los Sandboxes tienen una estructura de directorios estandarizada, donde cada directorio conserva una función específica dentro de la aplicación.

Cuando instala una aplicación en un dispositivo iOS, el sistema crea la estructura de directorios que se muestra en la siguiente figura.



Preferencias de usuario.

En la mayoría de aplicaciones podemos configurar una serie de parámetros para adaptarlo a las preferencias o necesidades del usuario. Son elementos como

colores, tipos de letra, unidades de medida, nombres de usuario y contraseñas de servicios de terceros.

iOS nos proporciona un mecanismo estándar para almacenar estas preferencias de forma permanente. La API es muy sencilla, permitiéndonos establecer valores por defecto, modificarlos según instrucciones del usuario y leer valores previamente establecidos.

Desde la perspectiva del tipo de datos, las preferencias del usuario no son más que una lista de propiedades, donde el objeto "raíz" es un NSDictionary. El sistema serializa/deserializa automáticamente los valores en el archivo .plist.

Un archivo .plist con preferencias se almacena automáticamente en el directorio Biblioteca/Preferencias de la aplicación. Su nombre será el nombre del proyecto Xcode, precedido por el "identificador de la organización" del proyecto.

De forma predeterminada, el .plist se genera en formato binario. Aunque no está en modo texto, aún podemos abrirlo y editarlo usando el editor .plist de Xcode. También podemos convertirlo a XML utilizando una herramienta de línea de comandos llamada Plutil, que se instala por defecto con Xcode.

Datos estructurados.

El almacenamiento de datos en una base de datos es ideal para datos estructurados o repetitivos, como la información de contacto.

Uno de los principios fundamentales de las bases de datos SQL es el esquema: una declaración formal de cómo se organiza una base de datos. El esquema se refleja en las instrucciones SQL que utiliza para crear la base de datos.

La clase de contrato es un contenedor de constantes que define nombres de URI, tablas y columnas. Esta clase le permite usar las mismas constantes en todas las demás clases en el mismo paquete, por lo que puede cambiar los nombres de las columnas en un solo lugar y propagar ese cambio a lo largo de su código.

Tipos de persistencia: local, remota y de Cacheo/Hoarding en dispositivos móviles.

Hay 3 maneras básicas de persistir la información las cuales son:

Preferencias.

Se utilizan parejas para una clave con su valor, además de tener las configuraciones, recordar acciones y más funciones del usuario.

Almacenamiento de archivos.

Se utiliza normalmente el almacenamiento interno del dispositivo y es aquí donde se guardan archivos arbitrarios tales como: imágenes, json, text, binarios, etc.

Datos estructurados.

Estas se refieren a las bases de datos SQL los cuales tienen colecciones de datos relacionados.

Dicho lo anterior la manera en la que se hace la persistencia en móvil puede ir desde local, remotamente o por cacheo.

Local.

Para la persistencia local se utiliza el almacenamiento propio del dispositivo, pero las desventajas que se tienen son: un espacio limitado, así como también que es de alto coste para las consultas complejas, aunque este tipo de persistencia es de acceso rápido.

Remota

Este se refiere a cuando hay una conexión remota con un servidor para consulta de datos y así se muestren. Desventajas de este tipo de persistencia es que cada acceso a datos requiere una consulta además de que puede haber alta latencia y muy probable a desconexiones.

Cacheo/Hoarding

Este se refiere cuando se tiene que hacer alguna copia local de información desde el servidor hacia el dispositivo. Para ello tiene que haber sincronización con el servidor para que sea funcional. Ventaja de este tipo es tener bajos índices de latencia.

Proceso de programación de persistencia en dispositivos móviles.

La API de DataStore a su vez tiene dos implementaciones diferentes:

- Preferencias DataStore: Esta es la forma más fácil de almacenar información utilizando pares clave-valor. No es necesario definir el esquema de datos de antemano, puedo guardar lo que quiera y no proporciona ningún tipo de coherencia al guardar la información. Puedo guardar String, Double o Float en la misma persistencia.

- Proto DataStore: En este caso, si se va a considerar el tipo a guardar, se debe definir el esquema antes de guardar los datos.

Para usar esta API, lo primero que tenemos que hacer es agregarla a nuestro proyecto, para esto solo necesitamos incluir en nuestro archivo Gradle:

Preferences DataStore.

```
// Preferences DataStore
    implementation "androidx.datastore:datastore-preferences:1.0.0-alpha01"


    // Proto DataStore
    implementation  "androidx.datastore:datastore-core:1.0.0-alpha01"
    implementation  "com.google.protobuf:protobuf-javalite:3.10.0"
```

Al comienzo del archivo buil.gradle de su aplicación, agregue:

```
plugins {
   id "com.google.protobuf" version "0.8.12"
}
```

Lo primero es crear un contenedor para las claves de las claves que queremos almacenar y los valores que queremos que persistan:

```
private val dataStore: DataStore<Preferences> = activity.createDataStore(
        name = "settings"
    )

companion object {
        val DAY_OF_MONTH = preferencesKey<Int>("day_of_month")
    }
```

Después de crear el contenedor, usaremos la clave creada para guardar, por ejemplo, el número 5.

```
suspend fun saveData(value: Int) {
        dataStore.edit { mutablePreferences: MutablePreferences ->
            mutablePreferences[DAY_OF_MONTH] = value
        }
    }
```

Vemos que es una función suspendida, y decíamos al principio que DataStore usa corrutinas y Flow para procesar datos de forma asíncrona.

Ahora si queremos restaurarlo, lo hemos guardado:

```
fun fetchData(): Flow<Int> {
        return dataStore.data.map { value: Preferences ->
            value[DAY_OF_MONTH] ?: 0
        }
    }
```

Mecanismos de tolerancia a fallos

Elementos para tomar en cuenta en el desarrollo de aplicaciones orientadas a móviles.

Uno de los factores que impulsan el éxito de una aplicación móvil es la optimización continua del rendimiento de extremo a extremo. Por lo tanto, si tiene contenido que

un usuario quiere o necesita, pero tarda más de 5 segundos en entregarlo, es probable que pierda a ese usuario debido al bajo rendimiento.

El rendimiento de la aplicación depende de los tiempos de inicio y carga, el tamaño de la aplicación, la velocidad de fotogramas, la compresión y más. La optimización de estos aspectos del rendimiento de la aplicación es fundamental para mejorar la experiencia del usuario lo más rápido posible, y los equipos pueden tener un impacto si se centran en algunas estrategias.

Consultas lentas.

Los usuarios tienen poca paciencia con los tiempos de inicio o carga lentos. Los desarrolladores deben asegurarse de que sus aplicaciones puedan ejecutarse en una variedad de redes para no tener problemas y no interrumpir su experiencia con las aplicaciones.

Sin embargo, hay buenas noticias: hay varias formas de mejorar inmediatamente el rendimiento de su aplicación:

1. Reduzca la cantidad de redireccionamientos de URL en la pantalla.
2. ¿Sigues usando Flash? ¡por! Encuentre una mejor alternativa compatible con dispositivos móviles.
3. Si el tiempo de respuesta del servidor backend es lento, su aplicación será lenta. La forma más fácil de evitar esto es evitar depender de servicios de hospedaje gratuitos o inapropiados que ofrecen poca o ninguna asistencia. La solución simple es invertir en servidores de alto rendimiento que eviten este tipo de problemas.
4. El mantenimiento de una base de datos nativa garantiza que, aunque el servidor falle, los datos del usuario no estén en peligro. Además, un servidor de respaldo garantiza que los usuarios puedan acceder a su aplicación incluso cuando el servidor principal está inactivo y mantiene las velocidades de inicio y carga que los usuarios esperan.

Alto consumo de batería.

La memoria y la duración de la batería son muy importantes para los usuarios móviles porque nuestros teléfonos tienen poco o ningún exceso. Por lo tanto, cuanto menos consumo de memoria y batería pueda causar una aplicación, mejor será tu experiencia de usuario.

Cosas para considerar:

1. Las fugas de memoria y las notificaciones automáticas son dos ejemplos que afectan el consumo de memoria.
2. El uso continuo de la aplicación puede agotar rápidamente la vida útil de la batería, así que asegúrese de considerar el consumo de energía innecesario durante la fase de desarrollo. Alternativamente, considere recordarles a los usuarios que apaguen funciones como GPS y Bluetooth cuando no estén en uso, lo que les ayudará a ahorrar batería y posicionar su aplicación como fácil de usar y para el consumidor.

Modo offline

Los usuarios son cada día más exigentes y no ven el modo fuera de línea como opcional, es una función que esperan que esté disponible

Nuestros usuarios siempre deben saber lo que sucede en la aplicación y, si se pierde la conexión, debemos notificarles qué sucede con el mensaje que acaban de enviar sin conexión o cuándo se actualizaron por última vez los datos que ven.

Un ejemplo: entramos en la aplicación de nuestra tienda de ropa favorita, vemos una prenda que nos gusta y hacemos un pedido fuera de línea según nuestra talla, que actualmente está en stock. Después de unas horas, nos volvimos a conectar e intentamos completar el pedido, pero se vendieron todas las existencias en nuestro tamaño. Debemos avisar a los usuarios.

Proceso de selección de los mecanismos de tolerancia a fallos en el desarrollo de aplicaciones de dispositivos móviles.

La tolerancia a fallas es la propiedad que permite que un sistema continúe funcionando normalmente si uno o más de sus componentes fallan.

Un sistema que tiene la capacidad interna de garantizar el funcionamiento correcto y continuo del sistema a pesar de las fallas de hardware o software.

Evite que se introduzcan fallas en el sistema antes de que entre en funcionamiento. Se utilizan en la fase de desarrollo del sistema.

Deje que el sistema siga funcionando, incluso en caso de falla. Se utilizan en la fase operativa del sistema. Es necesario conocer los posibles tipos de fallos, es decir, predecir fallos.

Solución de problemas y servicio continuo

• Estas cuatro etapas forman la base de todas las tecnologías tolerantes a fallas y deben existir en el diseño e implementación de sistemas tolerantes a fallas.

• Para proporcionar tolerancia a fallas, primero se deben detectar los efectos de los errores.

• El punto de partida de cualquier técnica de tolerancia a fallas es detectar estados defectuosos en la operación del sistema

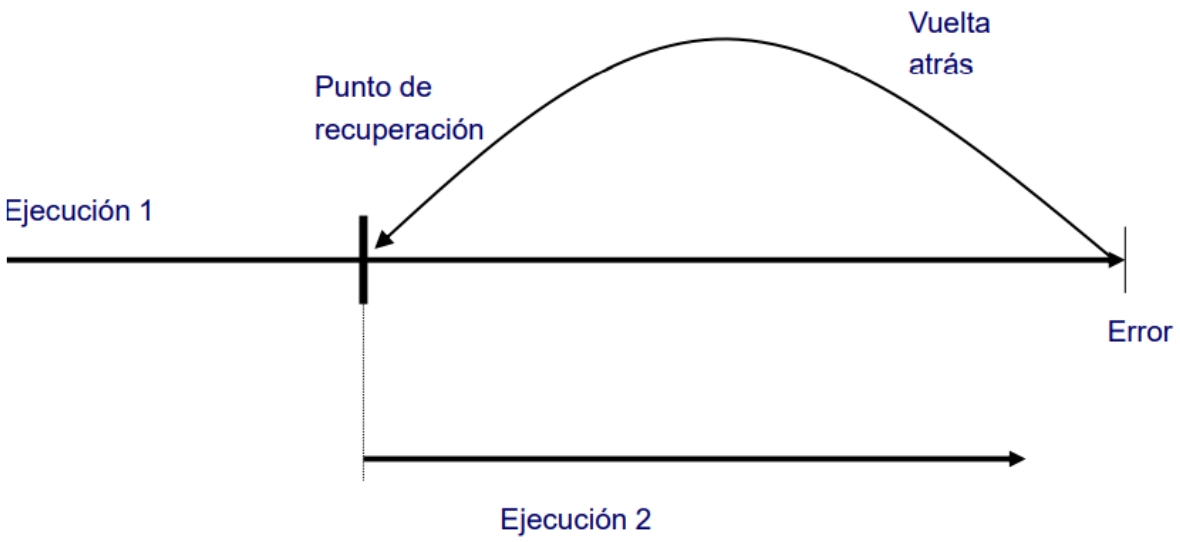
• Una vez que se detecta un error, el sistema debe recuperarse del error.

• Es necesario utilizar una técnica que convierta un estado de error del sistema en otro estado bien definido y libre de errores.

Dos técnicas básicas:

• Recuperar al revés. Cuando se detecta un error, el sistema vuelve al estado anterior sin errores.

• Punto de recuperación o punto de control, recuperación hacia adelante. Lleva el sistema a un estado libre de errores (sin retroceder).



Proceso de programación de los mecanismos de tolerancia a fallos en el desarrollo de aplicaciones de dispositivos móviles.

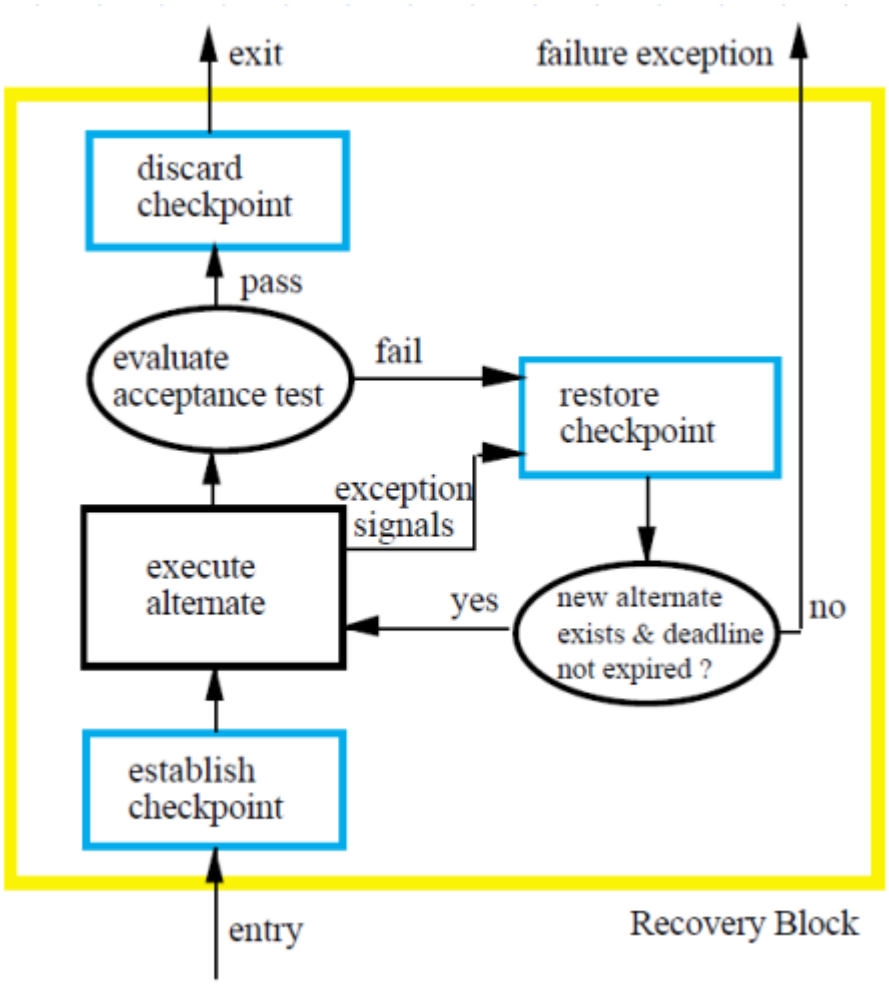
Casi todas las bases de datos (BD) incluyen una función de reversión, cuando un usuario realiza una acción en ella, se inicia una transacción, los cambios realizados durante este período no se fusionan inmediatamente en la base de datos, solo se actualizan después de que se completa la transacción y no se detecta un problema. Si la transacción falla, la base de datos no se actualiza.

bloque de recuperación

Es una técnica de recuperación hacia atrás integrada en los lenguajes de programación. Son bloques en el sentido normal de un lenguaje de programación, pero su entrada es un punto de recuperación, ya su salida se realiza una prueba de aceptación. Se utiliza para comprobar si finalmente el módulo principal del bloque se encuentra en un estado aceptable.

Si la prueba de aceptación falla, se restablece el estado inicial en el punto de recuperación, se ejecuta un módulo alternativo del mismo bloque, si vuelve a fallar, se continúan probando las alternativas, cuando ya no queda nada, el bloque falla y él se debe probar el siguiente bloque en un nivel más alto.

Esquema de recuperación



Posible sintaxis

Puede haber bloques anidados y si falla el bloque interior, se restaura el punto de recuperación del bloque exterior.

```
ensure <acceptance test>
by
    <primary module>
else by
    <alternative module>
else by
    <alternative module>
    ...
else by
    <alternative module>
else error
```

Conclusión

Las aplicaciones son importantes porque son impulsoras de la creatividad en sí mismas, también nos permiten comunicarnos con nuestro entorno y proporcionar entretenimiento o almacenar nuestras experiencias e información.

Son herramientas que permiten la interacción entre individuos, ya que permiten traducir el código en elementos visuales para actividades específicas. Las aplicaciones son importantes porque sin ellas no podríamos navegar por Internet, editar imágenes, escuchar música, comprar en línea y más.

Es por esto que, para lograr el correcto desarrollo de la aplicación, se deben considerar los temas que se muestran en la documentación. No sólo eso, sino que también vio su importancia.

Por ejemplo, si el medio de almacenamiento se encuentra en un entorno no controlado, la persistencia de los datos puede revelar información confidencial sin darse cuenta.

Bibliografía

Developers Android. (s.f.). Obtenido de Room: https://developer.android.com/training/data-storage/room?hl=es-419#java

ITPN. (s.f.). Obtenido de http://itpn.mx/recursositics/7semestre/desarrollodeaplicaciones/Unidad%20V.pdf

Richardmx. (27 de 05 de 2008). *Java Mexico*. Obtenido de QUE ES DAO: https://www.javamexico.org/blogs/richardmx/que_es_data_access_object