**-ANEXO**

1. Investigaciones
   1. Android
2. Qué es?
3. Comunidad
4. Arquitectura Andriod
5. Dalvik VM
   1. Diferencias con una máquina virtual Java normal
6. Aplicaciones de Android
   1. Componentes
   2. Estado de los procesos
   3. Ciclos de vida de una actividad
   4. Modelo de seguridad
7. Librerías
   1. Librerías básicas
   2. Librerías avanzadas
8. Interfaces de Usuario
9. Content Provider (SQLite)
   1. Seguridad en Base de Datos
10. Concepto
11. Medidas de seguridad
    1. Tipo de medidas
    2. Principales características
12. Requisitos para la seguridad de la base de datos
13. Servicios de seguridad
14. Identificación y autentificación
    1. Mecanismo de autentificación
    2. Matriz de autorización
15. Otras amenazas y ataques posibles
16. Inyección SQL
    1. Hibernate
17. Introducción
18. Que es Hibernate
19. Otras Características
    1. Testing
    2. Bibliografía

**1-Investigaciones:**

**1.1-Android:**

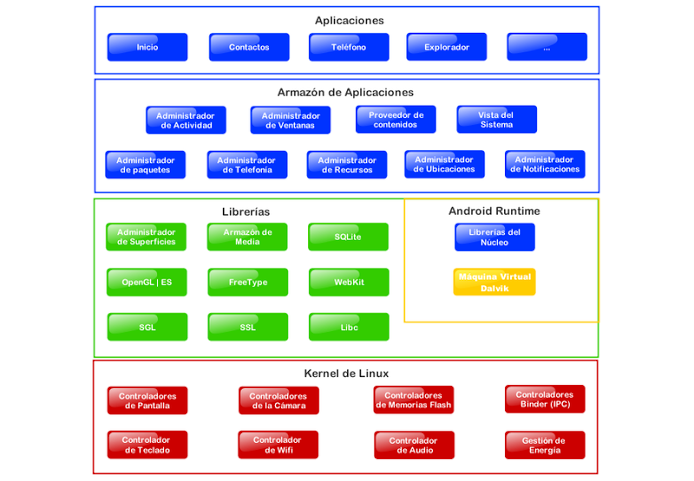
**1-Que es?:** Android es un sistema operativo basado en Linux propiedad de google. En un principio fue diseñado para dispositivos móviles con pantalla táctil, como teléfonos inteligentes o tablets. Actualmente también funciona en relojes inteligentes, televisores y automóviles.

Android inc fue la compañía de software que desarrollo inicialmente este sistema, luego obtuvo el respaldo de la empresa google la cual más tarde compraría Android en 2005. Años después fue presentado junto con la fundación Open Handser Alliance para avanzar en los estándares abiertos de los dispositivos móviles.

**2-Comunidad:** Tiene una gran comunidad de desarrolladores creando aplicaciones para extender la funcionalidad de los dispositivos. Actualmente se ha llegado a 1.000.000 de aplicaciones (de las cuales, dos tercios son gratuitas) disponibles en la tiendas oficial de Android Google Play, sin contar las aplicaciones de otras tiendas no oficiales de otras marcas.

### 3-Arquitectura Android: Como ya se ha mencionado, Android es una plataforma para dispositivos móviles que contiene una pila de software donde se incluye un sistema operativo, Middleware y aplicaciones básicas para el usuario.

### En las siguientes líneas se dará una visión global por capas de cuál es la arquitectura empleada en Android. Cada una de estas capas utiliza servicios ofrecidos por las anteriores, y ofrece a su vez los suyos propios a las capas de niveles superiores, tal como muestra la siguiente figura:



* **Aplicaciones:** Este nivel contiene, tanto las incluidas por defecto de Android como aquellas que el usuario vaya añadiendo posteriormente. Todas estas aplicaciones utilizan los servicios, las API y librerías de los niveles anteriores.
* **Framework de Aplicaciones:** Representa fundamentalmente el conjunto de herramientas de desarrollo de cualquier aplicación. Toda aplicación que se desarrolle para Android, ya sean las propias del dispositivo, las desarrolladas por Google o terceras compañías, o incluso las que el propio usuario cree, utilizan el mismo conjunto de API y el mismo "framework", representado por este nivel.

       Entre las API más importantes ubicadas aquí, se pueden encontrar las siguientes:

* **Activity Manager:** Conjunto de API que gestiona el ciclo de vida de las aplicaciones en Android.
* **Window Manager:** Gestiona las ventanas de las aplicaciones y utiliza la librería Surface Manager.
* **Telephone Manager:** Incluye todas las API vinculadas a las funcionalidades propias del teléfono (llamadas, mensajes, etc.).
* **Content Provider:** Permite a cualquier aplicación compartir sus datos con las demás aplicaciones de Android. Por ejemplo, gracias a esta API la información de contactos, agenda, mensajes, etc. será accesible para otras aplicaciones.
* **View System:** Proporciona un gran número de elementos para poder construir interfaces de usuario (GUI), como listas, mosaicos, botones, "check-boxes", tamaño de ventanas, control de las interfaces mediante teclado, etc. Incluye también algunas vistas estándar para las funcionalidades más frecuentes.
* **Location Manager:** Posibilita a las aplicaciones la obtención de información de localización y  posicionamiento.
* **Notification Manager:** Mediante el cual las aplicaciones, usando un mismo formato, comunican al usuario eventos que ocurran durante su ejecución: una llamada entrante, un mensaje recibido, conexión Wi-Fi disponible, ubicación en un punto determinado, etc. Si llevan asociada alguna acción, en Android denominada **Intent**, (por ejemplo, atender una llamada recibida) ésta se activa mediante un simple clic.
* **XMPP Service:** Colección de API para utilizar este protocolo de intercambio de mensajes basado en XML.
* **Librerías:**La siguiente capa se corresponde con las librerías utilizadas por Android. Éstas han sido escritas utilizando C/C++ y proporcionan a Android la mayor parte de sus capacidades más características. Junto al núcleo basado en Linux, estas librerías constituyen el corazón de Android.

              Entre las librerías más importantes ubicadas aquí, se pueden encontrar las siguientes:

* + **Librería libc:**Incluye todas las cabeceras y funciones según el estándar del lenguaje C. Todas las demás librerías se definen en este lenguaje.
  + **Librería Surface Manager:** Es la encargada de componer los diferentes elementos de navegación de pantalla. Gestiona también las ventanas pertenecientes a las distintas aplicaciones activas en cada momento.
  + **OpenGL/SL y SGL:** Representan las librerías gráficas y, por tanto, sustentan la capacidad gráfica de Android. OpenGL/SL maneja gráficos en 3D y permite utilizar, en caso de que esté disponible en el propio dispositivo móvil, el hardware encargado de proporcionar gráficos 3D. Por otro lado, SGL proporciona gráficos en 2D, por lo que será la librería más habitualmente utilizada por la mayoría de las aplicaciones. Una característica importante de la capacidad gráfica de Android es que es posible desarrollar aplicaciones que combinen gráficos en 3D y 2D.
  + **Librería Media Libraries:** Proporciona todos los códecs necesarios para el contenido multimedia soportado en Android (vídeo, audio, imágenes estáticas y animadas, etc.)
  + **FreeType:** Permite trabajar de forma rápida  y sencilla con distintos tipos de fuentes.
* **Librería SSL:** Posibilita la utilización de dicho protocolo para establecer comunicaciones seguras.
  + **Librería SQLite:**Creación y gestión de bases de datos relacionales.
  + **Librería WebKit:**Proporciona un motor para las aplicaciones de tipo navegador y forma el núcleo del actual navegador incluido por defecto en la plataforma Android.
* **Tiempo de ejecución de Android:**Al mismo nivel que las librerías de Android se sitúa el entorno de ejecución. Éste lo constituyen las **Core Libraries**, que son librerías con multitud de clases Java y la máquina virtual Dalvik.
* **Núcleo Linux**: Android utiliza el **núcleo de Linux 2.6** como una capa de abstracción para el hardware disponible en los dispositivos móviles. Esta capa contiene los drivers necesarios para que cualquier componente hardware pueda ser utilizado mediante las llamadas correspondientes. Siempre que un fabricante incluye un nuevo elemento de hardware, lo primero que se debe realizar para que pueda ser utilizado desde Android es crear las librerías de control o drivers necesarios dentro de este núcleo de Linux inserto en el propio Android.

**4-Dalvik VM** es el nombre de la máquina virtual que utiliza Android, la cual está basada en registro,  En ella podemos encontrar una gran diferencia con respecto a la máquina virtual Java (JVM), ya que la máquina virtual de Google no está basada en una pila (es una lista ordenada o [estructura de datos](http://es.wikipedia.org/wiki/Estructura_de_datos)).

Dalvik VM es un intérprete que sólo ejecuta los archivos ejecutables con formato .dex (*Dalvik Executable*). Este formato está optimizado para el almacenamiento eficiente de la memoria, lo cual consigue delegando en el núcleo la gestión de hilos, de memoria y de procesos.

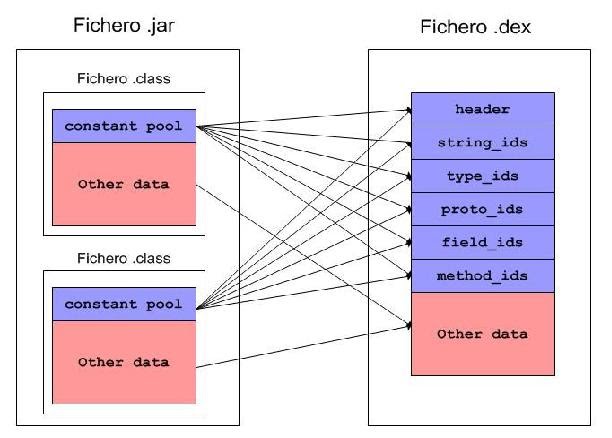
La herramienta "**dx**" incluida en el SDK de Android permite transformar las clases compiladas (.class) por un compilador de lenguaje Java en formato .dex.

**4.1-Diferencias con una máquina virtual Java normal:**

En primer lugar, la máquina virtual de Dalvik toma los archivos generados por las clases Java y los combina en uno o más archivos ejecutables Dalvik (. dex), los cuales a su vez son comprimidos en un sólo fichero **.apk** (*Android Package*) en el dispositivo. De esta forma, reutiliza la información duplicada por múltiples archivos .class, reduciendo así la necesidad de espacio (sin comprimir) a la mitad de lo que ocuparía un archivo .jar. En segundo lugar, Google ha mejorado la recolección de basura en la máquina virtual de Dalvik, pero ha preferido omitir **just-in-time (JIT)**. La empresa justifica esta elección diciendo que muchas de las bibliotecas centrales de Android, incluyendo las bibliotecas de gráficos, están implementadas en C y C++. Del mismo modo, Android proporciona una biblioteca de C optimizada para acceder a la base de datos **SQLite**, pero esta biblioteca está encapsulada en un nivel superior del API de Java. Dado que la mayoría del código del núcleo se encuentra en C y C++, Google argumentó que el impacto de la compilación JIT no sería significativo.

Por último, la máquina virtual de Dalvik utiliza un tipo diferente de montaje para la generación del código, en el que se utilizan los registros como las unidades primarias de almacenamiento de datos en lugar de la pila. Hay que señalar que el código ejecutable final de Android, como resultado de la máquina virtual de Dalvik, no se basa en el **bytecode** **de Java**, sino que se basa en los archivos .dex. Esto significa que no se puede ejecutar directamente el **Bytecode de Java,** sino que hay que comenzar con los archivos .class de Java y luego convertirlos en archivos .dex.

**Formato de un fichero .dex**

****

**5-Aplicaciones de Android:**

### Una aplicación Android corre dentro de su propio proceso Linux, por tanto, una característica fundamental de Android es que el tiempo y ciclo de vida de una aplicación no está controlado por la misma aplicación sino que lo determina el sistema a partir de una combinación de estados como pueden ser qué aplicaciones están funcionando, qué prioridad tienen para el usuario y cuánta memoria queda disponible en el sistema.

Una aplicación en Android debe declarar todas sus actividades, los puntos de entrada, la comunicación, las capas, los permisos, y las intenciones a través de **AndroidManifest.xml**. Es muy importante tener en consideración cómo estos componentes impactan en el tiempo de vida del proceso asociado con una aplicación, porque si no son empleados de manera apropiada, el sistema detendrá el proceso de la aplicación aun cuando se esté haciendo algo importante.

#### 5.1-Componentes:

* **Activity:** Sin duda es el componente más habitual de las aplicaciones para Android. Un componente Activity refleja una determinada actividad llevada a cabo por una aplicación, y que lleva asociada típicamente una ventana o interfaz de usuario; es importante señalar que no contempla únicamente el aspecto gráfico, sino que éste forma parte del componente Activity a través de vistas representadas por clases como View y sus derivadas. Este componente se implementa mediante la clase de mismo nombre Activity. La mayoría de las aplicaciones permiten la ejecución de varias acciones a través de la existencia de una o más pantallas. Por ejemplo, una aplicación de mensajes de texto. En ella, la lista de contactos se muestra en una ventana. Mediante el despliegue de una segunda ventana, el usuario puede escribir el mensaje al contacto elegido, y en otra tercera puede repasar su historial de mensajes enviados o recibidos. Cada una de estas ventanas debería estar representada a través de un componente Activity, de forma que navegar de una ventana a otra implica lanzar una actividad o terminar otra. Android permite controlar por completo el **ciclo de vida** de los componentes Activity. Tal y como se puede ver en el siguiente punto, una actividad tiene un ciclo de vida muy definido, que será igual para todas las actividades. Este ciclo de vida es impuesto por el SDK de Android. Las actividades tienen cuatro posibles estados:**Activa, pausada, parada y reiniciada.**

A la hora de diseñar una aplicación en Android hay que tener en cuenta su ciclo de vida.

* **Intent:**Un Intent consiste básicamente en la voluntad de realizar alguna acción, generalmente asociada a unos datos. Lanzando un Intent, una aplicación puede delegar el trabajo en otra, de forma que el sistema se encarga de buscar qué aplicación de entre las instaladas, es la que puede llevar a cabo la acción solicitada. Por ejemplo, abrir una URL en algún navegador web, o escribir un correo electrónico desde algún cliente de correo. Los Intents están incluidos en el **AndroidManifest** porque describen dónde y cuándo puede comenzar una actividad. Cuando una actividad crea un Intent, éste puede tener descriptores de lo que se quiere hacer. Una vez se está ejecutando la aplicación, Android compara esta información del Intent con los Intents de cada aplicación, eligiendo el más adecuado para realizar la operación especificada por el llamante.
* **Broadcast Intent Receiver:**Un componente Broadcast Intent Receiver se utiliza para lanzar alguna ejecución dentro de la aplicación actual cuando un determinado evento se produzca (generalmente, abrir un componente Activity*)*. Por ejemplo, una llamada entrante o un SMS recibido. Este componente no tiene interfaz de usuario asociada, pero puede utilizar el API *Notification Manager*para avisar al usuario del evento producido a través de la barra de estado del dispositivo móvil. Este componente se implementa a través de una clase de nombre BroadcastReceiver. Para queBroadcast Intent Receiver funcione, no es necesario que la aplicación en cuestión sea la aplicación activa en el momento de producirse el evento. El sistema lanzará la aplicación si es necesario cuando el evento tenga lugar.
* **Service:**Un componente Service representa una aplicación ejecutada sin interfaz de usuario, y que generalmente tiene lugar en segundo plano mientras otras aplicaciones (éstas con interfaz) son las que están activas en la pantalla del dispositivo. Un ejemplo típico de este componente es un reproductor de música. La interfaz del reproductor muestra al usuario las distintas canciones disponibles, así como los típicos botones de reproducción, pausa, volumen, etc. En el momento en el que el usuario reproduce una canción, ésta se escucha mientras se siguen visualizando todas las acciones anteriores, e incluso puede ejecutar una aplicación distinta sin que la música deje de sonar. La interfaz de usuario del reproductor sería un componente Activity, pero la música en reproducción sería un componente Service, porque se ejecuta en *background*. Este elemento está implementado por la clase de mismo nombre Service.
* **Content Provider:** Con el componente Content Provider, cualquier aplicación en Android puede almacenar datos en un fichero, en una base de datos SQLite o en cualquier otro formato que considere. Además, estos datos pueden ser compartidos entre distintas aplicaciones. Una clase que implemente el componente Content Provider contendrá una serie de métodos que permiten almacenar, recuperar, actualizar y compartir los datos de una aplicación.

Existe una colección de clases para distintos tipos de gestión de datos en el paquete **android.provider***.* Además, cualquier formato adicional que se quiera implementar deberá pertenecer a este paquete y seguir sus estándares de funcionamiento.

No todas las aplicaciones tienen que tener los cuatro componentes, pero cualquier aplicación será una combinación de estos.

**5.2-Estado de los procesos:**

Como se ha mencionado anteriormente, cada **aplicación** de Android corre en su **propio proceso**, el cual es creado por la aplicación cuando se ejecuta y permanece hasta que la aplicación deja de trabajar o el sistema necesita memoria para otras aplicaciones. Android sitúa cada proceso en una jerarquía de "importancia" basada en estados, como se puede ver a continuación:

* **Procesos en primer plano (*Active process*):** Es un proceso que aloja una Activity en la pantalla y con la que el usuario está interactuando (su método **onResume()** ha sido llamado) o que un **IntentReceiver** está ejecutándose. Este tipo de procesos serán eliminados como último recurso si el sistema necesitase memoria.

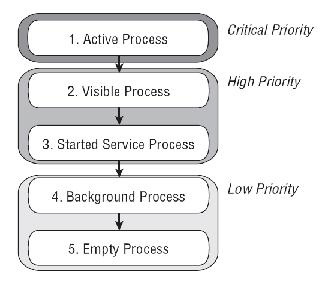
* **Procesos visibles (*Visible process*)**: Es un proceso que aloja una Activity pero no está en primer plano (su métodoonPause() ha sido llamado). Esto ocurre en situaciones dónde la aplicación muestra un cuadro de diálogo para interactuar con el usuario. Este tipo de procesos no será eliminado en caso que sea necesaria la memoria para mantener a todos los procesos del primer plano corriendo.

* **Procesos de servicio (*Started service process*)**: Es un proceso que aloja un Service que ha sido iniciado con el método startService(). Este tipo de procesos no son visibles y suelen ser importantes para el usuario (conexión con servidores, reproducción de música).

* **Procesos en segundo plano (*Background process*)**: Es un proceso que aloja una Activity que no es actualmente visible para el usuario (su métodoonStop() ha sido llamado). Normalmente la eliminación de estos procesos no suponen un gran impacto para la actividad del usuario. Es muy usual que existan numerosos procesos de este tipo en el sistema, por lo que el sistema mantiene una lista para asegurar que el último proceso visto por el usuario sea el último en eliminarse en caso de necesitar memoria.

* **Procesos vacíos (*Empty process*)**: Es un proceso que no aloja ningún componente. La razón de existir de este proceso es tener una caché disponible de la aplicación para su próxima activación. Es común, que el sistema elimine este tipo de procesos con frecuencia para obtener memoria disponible.

Según esta jerarquía, Android prioriza los procesos existentes en el sistema y decide cuáles han de ser eliminados, con el fin de liberar recursos y poder lanzar la aplicación requerida.



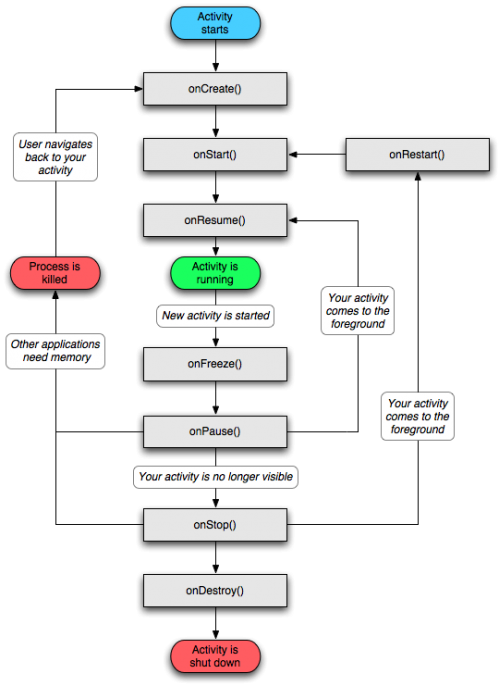
Para los procesos en segundo plano, existe una lista llamada **LRU**(*Least Recently Used*). En función de esta lista se van eliminando los procesos; los primeros que se eliminan son aquellos que llevan más tiempo sin usarse. Así el sistema se asegura de mantener vivos los procesos que se han usado recientemente.

**5.3-Ciclos de vida de una actividad:**

El hecho de que cada aplicación se ejecuta en su propio proceso aporta beneficios en cuestiones básicas como seguridad, gestión de memoria, o la ocupación de la CPU del dispositivo móvil. Android se ocupa de lanzar y parar todos estos procesos, gestionar su ejecución y decidir qué hacer en función de los recursos disponibles y de las órdenes dadas por el usuario. El usuario desconoce este comportamiento de Android. Simplemente es consciente de que mediante un simple clic o toque pasa de una a otra aplicación y puede volver a cualquiera de ellas en el momento que lo desee. No debe preocuparse sobre cuál es la aplicación que realmente está activa, cuánta memoria está consumiendo, ni si existen o no recursos suficientes para abrir una aplicación adicional. Todo eso son tareas propias del sistema operativo. Android lanza tantos procesos como permitan los recursos del dispositivo. Cada proceso, correspondiente a una aplicación, estará formado por una o varias actividades independientes (componentes Activity) de esa aplicación. Cuando el usuario navega de una actividad a otra, o abre una nueva aplicación, el sistema duerme dicho proceso y realiza una copia de su estado para poder recuperarlo más tarde. El proceso y la actividad siguen existiendo en el sistema, pero están dormidos y su estado ha sido guardado. Es entonces cuando crea, o despierta si ya existe, el proceso para la aplicación que debe ser lanzada, asumiendo que existan recursos para ello.

Cada uno de los componentes básicos de Android tiene un ciclo de vida bien definido; esto implica que el desarrollador puede controlar en cada momento en qué estado se encuentra dicho componente, pudiendo así programar las acciones que mejor convengan.

El componente**Activity,** probablemente el más importante, tiene un ciclo de vida como el mostrado en la siguiente figura.



De la figura anterior, pueden sacarse las siguientes conclusiones:

* **onCreate(), onDestroy():**Abarcan todo el ciclo de vida. Cada uno de estos métodos representan el principio y el fin de la actividad.
* **onStart(), onStop():**Representan la parte visible del ciclo de vida. Desde onStart() hasta onStop(), la actividad será visible para el usuario, aunque es posible que no tenga el foco de acción por existir otras actividades superpuestas con las que el usuario está interactuando. Pueden ser llamados múltiples veces.
* **onResume(), onPause():** Delimitan la parte útil del ciclo de vida. Desde onResume() hasta onPause(), la actividad no sólo es visible, sino que además tiene el foco de la acción y el usuario puede interactuar con ella.

Tal y como se ve en el diagrama de la figura anterior, el proceso que mantiene a esta Activity puede ser eliminado cuando se encuentra en onPause()o enonStop()*,* es decir, cuando no tiene el foco de la aplicación. Android nunca elimina procesos con los que el usuario está interactuando en ese momento. Una vez se elimina el proceso, el usuario desconoce dicha situación y puede incluso volver atrás y querer usarlo de nuevo. Entonces el proceso se restaura gracias a una copia y vuelve a estar activo como si no hubiera sido eliminado. Además, la Activity puede haber estado en segundo plano, invisible, y entonces es despertada pasando por el estado onRestart()*.* Pero, ¿qué ocurre en realidad cuando no existen recursos suficientes? Obviamente, los recursos son siempre limitados, más aun cuando se está hablando de dispositivos móviles. En el momento en el que Android detecta que no hay los recursos necesarios para poder lanzar una nueva aplicación, analiza los procesos existentes en ese momento y elimina los procesos que sean menos prioritarios para poder liberar sus recursos. Cuando el usuario regresa a una actividad que está dormida, el sistema simplemente la despierta. En este caso, no es necesario recuperar el estado guardado porque el proceso todavía existe y mantiene el mismo estado. Sin embargo, cuando el usuario quiere regresar a una aplicación cuyo proceso ya no existe porque se necesitaba liberar sus recursos, Android lo crea de nuevo y utiliza el estado previamente guardado para poder restaurar una copia fresca del mismo.

**5.4-Modelo de seguridad:**

### La mayoría de las medidas de seguridad entre el sistema y las aplicaciones derivan de los estándares de Linux 2.6, cuyo núcleo, constituye el núcleo principal de Android.

Cada proceso en Android proporciona  un  entorno  seguro  de  ejecución, en el que por  defecto,  ninguna  aplicación  tienepermiso  para  realizar  ninguna  operación  o   comportamiento  que  pueda  impactar negativamente en la ejecución de otras aplicaciones o del sistema mismo. Por ejemplo, acciones como  leer o escribir carpetas o documentos privados del usuario no  están permitidas.  La  única  forma  de  poder saltar estas restricciones impuestas por Android, es mediante un permiso que autorice a llevar a cabo una determinada acción prohibida. La seguridad de Android abarca desde el despliegue hasta la ejecución de la aplicación.

Con respecto a la ejecución, cada aplicación Android se ejecuta dentro de un proceso separado. Cada uno  de  estos procesos  tiene  un  ID  de  usuario  único  y  permanente  (asignado  en  el momento de  la  instalación). Esto  impone un límite  al proceso porque  evita que una aplicación tenga acceso directo a los datos críticos para el correcto funcionamiento del terminal.

Para establecer un permiso para una aplicación, es necesario declarar uno o más elementos**<uses-permission>** donde se especifica el tipo de permiso que se  desea  habilitar. Por ejemplo, si se quisiera permitir que una aplicación pueda monitorizar mensajes SMS entrantes, en el fichero **AndroidManifest.xml** se encontraría algo como lo que sigue:

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

          package="com.android.app.myapp"

<uses-permission android:name="**android.permission.RECEIVE\_SMS**" />

    </manifest>

En la clase **"android.Manifest.Permission"** se especifican todos los posibles permisos que se pueden conceder a una aplicación.

El elemento **<uses-permission>** contempla una serie de atributos que definen el alcance el permiso dado:

* **android:name:**Especificación del permiso que se pretende conceder. Debe ser un nombre de alguno de los listados en la clase **android.Manifest.Permission**.

* **android:label:** Una etiqueta o nombre convencional fácilmente legible para el usuario.

* **android:permissionGroup:** Permite  especificar  un  grupo  asociado al permiso. Los posibles grupos están en **android.Manifest.permission\_group.**

* **android:protectionLevel:** Determina  el nivel de  riesgo del permiso, y en función del mismo influye en como el sistema otorga o no el permiso a la aplicación. Los valores son entre 0 y 3.

* **android:description:**Descripción textual del permiso.

* **android:icon:** Icono gráfico que puede ser asociado al permiso.

**6-Librerías**

**6.1-Librerias Básicas:**

Android ofrece una serie de **APIs** para el desarrollo de sus **aplicaciones**. La siguiente lista proporciona una visión de lo que está disponible. Todos los dispositivos Android ofrecen al menos estas APIs:

* **android.util:** El paquete básico de servicios públicos contiene las clases de bajo nivel, como contenedores especializados, formateadores de cadenas, y de análisis XML de servicios públicos.
* **android.os:** El paquete de sistema operativo permite el acceso a los servicios básicos como el paso de mensajes, la comunicación entre procesos y  funciones de reloj.
* **android.graphics:** La API de gráficos, es el suministro de las clases de bajo nivel como colores y las primitivas de dibujo.
* **android.text:** Las herramientas de procesamiento de texto para mostrarlo y analizarlo.
* **android.database:** Proporciona las clases de bajo nivel necesario para la manipulación de cursores cuando se trabaja con bases de datos.
* **android.content:** El contenido de la API se utiliza para admirar el acceso a los datos y a la publicación, proporcionando los servicios para hacer frente a los recursos, los proveedores de contenido y los paquetes.
* **android.view:**Las vistas son un núcleo de la interfaz de usuario. Todos los elementos de la interfaz se construyen utilizando una serie de vistas que proporcionan los componentes de interacción con el usuario.
* **android.widget:** Construido sobre el paquete de Vista, elementos de la interfaz de usuario para su uso en las aplicaciones. Se incluyen listas, botones y diseños.
* **android.app:**Paquete de alto nivel que proporciona el acceso al modelo de solicitud. Este,  incluye la actividad de servicios y las API que forman la base de todas sus aplicaciones.
* **com.google.android.maps:** API de alto nivel que proporciona acceso a los controles de mapas. Incluye el control MapView así como la clase MapController utilizados para anotar y controlar dichos mapas.
* **android.provider:** Para facilitar el acceso a los desarrolladores a determinados proveedores de contenidos estándar, el paquete proveedor ofrece clases para todas sus distribuciones.
* **android.telephony:** Las API´s de telefonía le dan la posibilidad de interactuar directamente con el dispositivo de Teléfono, permitiéndole realizar, recibir y controlar las llamadas de teléfono, su estado y mensajes SMS.
* **android.webkit:** Ofrece funciones para trabajar con contenido basado en web, incluyendo un control WebView para insertar los navegadores en sus actividades y un administrador de cookies.

**6.2*-*Librerías Avanzadas:**

Android pretende abarcar una gran variedad de equipos móviles, de modo que tenga en cuenta que la adecuación y aplicación de las siguientes API´s variará en función del dispositivo a que se apliquen:

* **android.location**: Da las aplicaciones de acceso a la ubicación física del dispositivo actual. Los servicios de localización ofrecen acceso genérico a información de localización utilizando cualquier posición de hardware xing-Fi o tecnología disponible en el dispositivo.
* **android.media:** Las API de los medios de comunicación proporcionan soporte para reproducción y grabación de audio.
* **android.opengl:** Android ofrece un potente motor 3D que utiliza la API de OpenGL ES. Sirve para las interfaces de usuarios en 3D.
* **android.hardware:** Cuando sea posible, el hardware de la API expone un sensor incluyendo la cámara, acelerómetro, sensores y brújula.
* **android.bluetooth, android.net.wiﬁ, android and telephony:** Proporciona el acceso a las plataformas hardware, incluyendo Bluetooth, Wi-Fi y el hardware de telefonía.

### 7-Interfaces de Usuario:

### La interfaz de usuario se define en los archivos XML del directorio res/layout. Cada pantalla tendrá un código XML diferente.

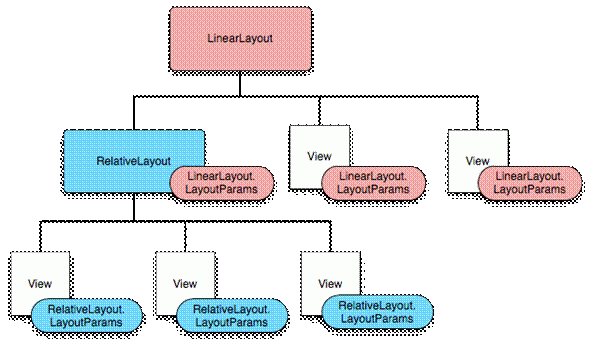
Diseñar una pantalla usando Java puede resultar complejo y poco eficiente, sin embargo, Android soporta XML para diseñar pantallas y define elementos personalizados, cada uno representando a un "subclase" especifica de view. Se pueden crear pantallas de la misma manera que se diseñan ficheros HTML.

Cada fichero describe un layout(una pantalla) y cada layout a su vez puede contener otros elementos. Para gestionar la interfaz de usuario, Android introduce las siguientes terminologías:

**View**: Una view es un objeto cuya clase es **android.view.View**. Es una estructura de datos cuyas propiedades contienen los datos de la capa, la información específica del área rectangular de la pantalla y permite establecer el layout. Una view tiene: layout, drawing, focus change, scrolling, etc..La clase view es útil como clase base para los widgets, que son unas subclases ya implementadas que dibujan los elementos en la pantalla. Los widgets contienen sus propias medidas, pero se pueden usar para construir una interfaz más rápidamente. La lista de widgets que se pueden utilizar incluye Text, EditText, InputMethod, MovementMethod, Button, RadioButton, CheckBox, y ScrollView.

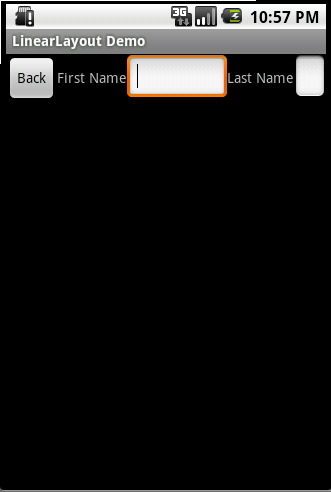
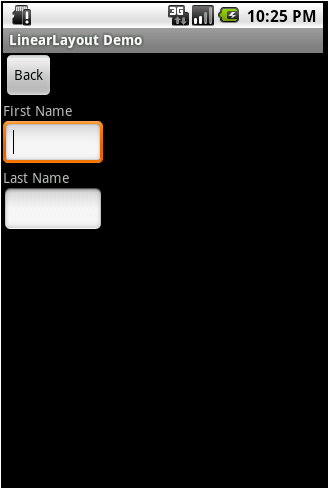
**Viewgroups**: Un viewgroup es un objeto de la clase **android.view.Viewgroup**, como su propio nombre indica, un viewgroup es un objeto especial de view cuya funcion es contener y controlar la lista de views y de otros viewgroups. Los viewgroups permiten añadir estructuras a la interfaz y acumular complejos elementos en la pantalla que son diseccionados por una sola entidad.La clase viewgroup es útil como base de la clase layouts, que son subclases implementadas que proveen los tipos más comunes de los layouts de pantalla. Los layouts proporcionan una manera de construir una estructura para una lista de views.

**Árbol estructurado de la interfaz UI:**En la plataforma Android se define una Activity del UI usando un árbol de nodos view y viewgroups, como vemos en la imágen de abajo. El árbol puede ser tan simple o complejo como se necesite hacerlo, y se puede desarrollar usando los widgets y layouts que Android proporciona o creando views propias

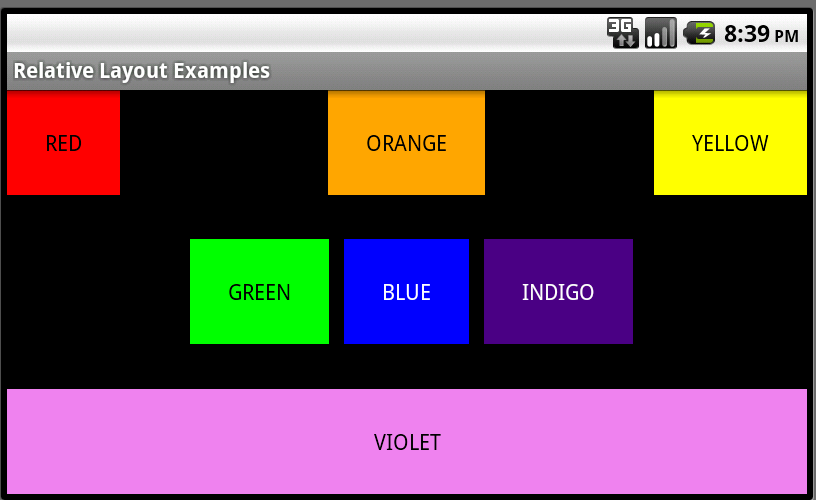


Los views y viewgroups deben estar contenidos en los layouts, los cuales contienen otros elementos presentes en una vista. Dentro de cada layout podemos poner todos los elementos necesarios, incluidos otros layouts. Así conseguiremos estructurar la pantalla de la manera deseada. Existen una gran variedad de layouts, en función de su posicionamiento en la pantalla y se describen a continuación algunos de ellos:

**LinearLayout**: Se les conoce como contenedores y sirven para reorganizar los elementos de nuestra aplicación. Sus hijos son los views, viewgroup u otros layouts. Nos permitirán alinear sus hijos en una única dirección, ya sea horizontal o vertical como se muestra en las siguientes imágenes. La orientación predeterminada es horizontal.

**RelativeLayout**: En este caso todos los elementos van colocados en una posición relativa a otro. Aquí podemos jugar con las distancias entre elementos en la pantalla, la cual se expresa en pixeles. A continuación se muestra un ejemplo:



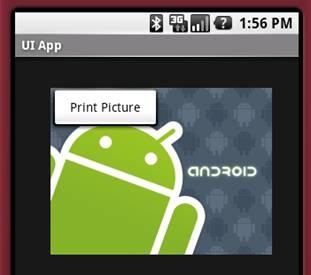
**Absolute layout**: Coloca los elementos en posiciones absolutas en la pantalla, teniendo en cuenta que la posición (0,0) es el extremo superior izquierdo de la pantalla.



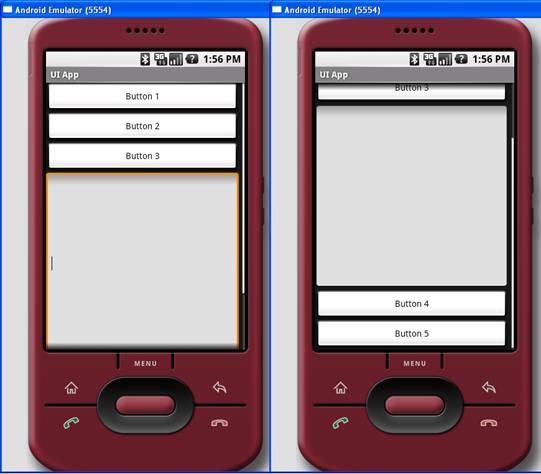
**TableLayout**: permite colocar los elementos en forma de tabla. Se utiliza el elemento <TableRow> para designar a una fila de la tabla. Cada fila puede tener uno o más puntos de vista. Cada vista se coloca dentro de una fila en forma de celda.



**FrameLayout**: Es un marcador de posición que puede usarse para mostrar una única vista. Se pueden agregar múltiples puntos de vista a un FrameLayout pero cada uno se acumulará en la parte superior izquierda de la anterior. A continuación mostramos un ejemplo ilustrativo:



**ScrollView**: Es un tipo especial de FrameLayout ya que permite a los usuarios desplazarse por una serie de puntos de vista que ocupan más espacio que el despliegue físico. El scrollView solo puede contener un viewGroup y suele ser LinearLayout.



**8-Content Provider (SQLite):**

Content Provider es una superclase para el manejo y almacenamiento de datos**.**Es la manera de **compartir** **datos** entre **procesos y aplicaciones**, ya que los datos de cada aplicación se almacenan por separado.

Una clase de Content Providerimplementa un conjunto estándar de métodos para exponer sus datos de aplicación para ser utilizados, consumidos o creados por otras aplicaciones. Normalmente, ContentProvider se usa con una **base de datos** (SQLite), pero puede usarse con otro medio de compartición de variables.

Por defecto, Android incluye una serie de componentes Content Provider que permiten publicar todo tipo de datos básicos que pueden resultar útiles entre aplicaciones: información de los contactos, fotografías, imágenes, vídeos, mensajes de texto, audios, etc. Todos estos Content Provider ya definidos y listos para utilizar se pueden encontrar en el paquete: **android.provider**.

Además, Android ofrece la posibilidad de que el desarrollador pueda crear sus propios Content Provider. Un Content Provider es un objeto de la clase ContentProvider, ubicada en el paquete **android.content**, y que almacena datos de un determinado tipo que pueden ser accedidos desde cualquier aplicación. Cada elemento Content Provider tiene asociada una **URI** única que lo representa y a través de la cual los otros componentes de una aplicación pueden acceder a él.

La información que se quiere obtener es el ID del contacto, su nombre y su teléfono. Para hacer una consulta a través de una URI, existen varios métodos disponibles: uno de ellos es **managedQuery()**, de la clase **Activity**. Este método requiere como parámetros la URI que se consulta, los campos a seleccionar, y otros valores como las cláusulas WHERE u ORDER BY, como si de una consulta SQL se tratara. El resultado es en un conjunto de filas que se puede recorrer a través del objeto **Cursor**, clase ubicada en el paquete **android.database.**

// Columnas a consultar

String[] projection = **new**String[] {

Contacts.People.*NAME*,

Contacts.People.*NUMBER*,

Contacts.People.*\_ID*

};

// Establecer URI para acceder a los contactos

Uri contacts = "content://contacts/people/";

// Lanzar consulta

Cursor cursor = managedQuery( contacts, projection, **null**, **null**, Contacts.People.*NAME*+ " ASC");

**1.2-Seguridad en Base de Datos:**

**1-Concepto:** Además de seguridad también se le puede llamar privacidad. El problema de la seguridad consiste en lograr que los recursos de un sistema sean, bajo toda circunstancia, utilizados para los fines previstos.

El concepto de seguridad lo podemos medir en:

* La protección del sistema frente a ataques externos.
* La protección frente a caídas o fallos en el software o en el equipo.
* La protección frente a manipulación por parte de usuarios no autorizados.

La seguridad de los datos se refiere a la protección de estos contra el acceso por parte de las personas no autorizadas y contra su indebida destrucción o alteración.

El analista de sistemas que se hace responsable de la seguridad debe estar familiarizado con todas las particularidades del sistema, porque este puede ser atacado con fines ilícitos desde muchos ángulos. A veces se presta mucha atención a alguno de los aspectos del problema mientras se descuidan otros.

**2-Medidas de Seguridad:** Existen una serie de medidas ya conocidas imprescindibles para la seguridad en base de datos. Las mismas son a diferentes niveles:

* **Físicas**: Controlar el acceso al equipo. Esto se logra mediante tarjetas de acceso, huellas digitales, etc.
* **Personal**: Acceso solo de personal autorizado. Para esto se debe lograr una confianza a la persona que tenga acceso a los diferentes datos.
* **SO**: Seguridad a nivel de SO.
* **SGBD**: Uso herramientas de seguridad que proporcione el SGBD. Perfiles de usuario, vistas, restricciones de uso de vistas, etc.
* **Autorización**: usar  derechos  de  acceso  dados  por el terminal,  por  la operación que puede realizar o por la hora del día.
* **Uso de técnicas de cifrado**: para proteger datos en Base de Datos distribuidas o con acceso por red o internet.
* **Diferentes tipos de cuentas**: permisos  para: creación de cuentas, agregado y borrado de privilegios y asignación de los niveles de seguridad.
* **Manejo de la tabla de usuarios** con código y contraseña, control de las operaciones efectuadas en cada sesión de trabajo por cada usuario y anotadas en algún documento, lo cual facilita la auditoría de la Base de Datos.

**2.1-Tipo de Medidas:**

**A-Discrecional:**  Se  usa para agregar  y  eliminar privilegios  a  los  usuarios  a  nivel  de archivos, registros o campos en un modo determinado (consulta o modificación).

**B-Obligatoria:** sirve para imponer seguridad de varios niveles tanto para los usuarios como para los datos.

Los sistemas operativos proveen algunos mecanismos de protección para poder implementar políticas de seguridad. Las políticas definen qué hay que hacer (qué datos y recursos deben protegerse de quién; es un problema de administración), y los mecanismos determinan cómo hay que hacerlo. Esta separación es importante en términos de flexibilidad, puesto que las políticas pueden variar en el tiempo y de una organización a otra. Los mismos mecanismos, si son flexibles, pueden usarse para implementar distintas políticas.

Un aspecto importante de la seguridad es el de impedir la pérdida de información, la cual puede producirse por diversas causas: fenómenos naturales, guerras, errores de hardware o de software, o errores humanos. La solución es una sola: mantener la información respaldada, de preferencia en un lugar lejano.

**2.2-Principales Características:**

Las tres principales características que se deben mantener en una base de datos son **la confidencialidad, la integridad y la disponibilidad de la información**. Los datos contenidos en una Base de Datos pueden ser individuales o de una Organización. Sean de un tipo o de otro, a no ser que su propietario lo autorice, no deben ser revelados. Si esta revelación es autorizada por dicho propietario la confidencialidad se mantiene. Es decir, asegurar la confidencialidad significa prevenir/ detectar/ impedir la revelación impropia de la información.

**3-Requisitos para la seguridad de la Base de Datos:**

* La base de datos debe ser protegida contra el fuego, el robo y otras formas de destrucción.
* Los datos deben ser re construibles, porque por muchas precauciones que se tomen, siempre ocurren accidentes.
* Los datos deben poder ser sometidos a procesos de auditoria. La falta de auditoria en los sistemas de computación ha permitido la comisión de grandes delitos.
* El sistema debe diseñarse a prueba de intromisiones. Los programadores, por ingeniosos que sean, no deben poder pasar por alto los controles.
* Ningún sistema puede evitar de manera absoluta las intromisiones malintencionadas, pero es posible hacer que resulte muy difícil eludir los controles.
* El sistema debe tener capacidad para verificar que sus acciones han sido autorizadas.
* Las acciones de los usuarios deben ser supervisadas, de modo tal que pueda descubrirse cualquier acción indebida o errónea.

**4-Servicios de Seguridad:**

Existen varios servicios y tecnologías relacionadas con la seguridad. Algunas de ellas son:

**Autenticación:** Se examinan las capacidades de login único a la red, autenticación y seguridad. Además, se proporciona información sobre el interfaz Security Support Provider Interface (SSPI) para obtener servicios de seguridad integrados del sistema operativo.

**Sistema de Archivos Encriptado**: El Sistema de Archivos Encriptado (Encrypted File System - EFS) proporciona la tecnología principal de encriptación de archivos para almacenar archivos del sistema de archivos NTFS de Windows NT encriptados en disco.

**Seguridad IP:** Windows IP Security, del Internet Engineering Task Force, proporciona a los administradores de redes un elemento estratégico de defensa para la protección de sus redes.

**Tarjetas Inteligentes:** se examinan los procesos de autenticación utilizando tarjetas inteligentes y los protocolos, servicios y especificaciones asociadas.

**Tecnologías de Clave Pública:** se revisa la infraestructura de clave pública incluida en los sistemas operativos de Microsoft y se proporciona información sobre criptografía.

**Identificar y autorizar a los usuarios:** uso de códigos de acceso y palabras claves, exámenes, impresiones digitales, reconocimiento de voz, barrido de la retina, etc.

**Autorización:** usar derechos de acceso dados por el terminal, por la operación que puede realizar o por la hora del día.

**Uso de técnicas de cifrado:** para proteger datos en BD distribuidas o con acceso por red o internet.

**Diferentes tipos de cuentas:** con permisos para: creación de cuentas, concesión y revocación de privilegios y asignación de los niveles de seguridad. Manejo de la tabla de usuarios con código y contraseña, control de las operaciones efectuadas en cada sesión de trabajo por cada usuario y anotadas en la bitácora, lo cual facilita la auditoría de la BD.

**5-Identificación y autentificación:**

En primer lugar el sistema debe identificar y autentificar a los usuarios utilizando alguno de las siguientes formas:

* Código y contraseña
* Identificación por hardware
* Conocimiento, aptitudes y hábitos del usuario
* Información predefinida (Aficiones, cultura, etc)

Además, el administrador deberá especificar los privilegios que un usuario tiene sobre los objetos:

* Usar una B.D.
* Consultar ciertos datos
* Actualizar datos
* Crear o actualizar objetos
* Ejecutar procedimientos almacenados
* Referenciar objetos
* Indexar objetos
* Crear identificadores

**5.1-Mecanismos de autentificación:**

La autentificación, que consiste en identificar a los usuarios que entran al sistema, se puede basar en posesión (llave o tarjeta), conocimiento (clave) o en un atributo del usuario (huella digital).

**-Claves**: El mecanismo de autentificación más ampliamente usado se basa en el uso de claves o passwords; es fácil de entender y fácil de implementar. Con claves de 7 caracteres tomados al azar de entre los 95 caracteres ASCII que  se pueden digitar con cualquier teclado, entonces las 957 posibles claves deberían desincentivar cualquier  intento por adivinarla. Sin embargo, una proporción demasiado grande de las claves escogidas por los usuarios son fáciles de adivinar,  pues la  idea es  que  sean  también  fáciles de recordar. La clave también se puede descubrir mirando (o filmando)  cuando  el  usuario  la  digita,  o si el  usuario hace login remoto, interviniendo la red y observando todos los paquetes que pasan por ella. Por último, además de que las claves se pueden descubrir, éstas también se pueden "compartir", violando las reglas de seguridad. En definitiva, el  sistema no tiene ninguna garantía de que quien hizo login es realmente el usuario que se supone que es.

**-Identificación física**: Un enfoque diferente es usar un elemento físico difícil de copiar, típicamente  una  tarjeta con una banda magnética. Para mayor seguridad este enfoque se suele combinar con una clave (como es el caso de los cajeros automáticos). Otra posibilidad es medir características físicas  particulares del sujeto: huella digital, patrón de vasos sanguíneos de la retina, longitud de los dedos. Incluso la firma sirve.

**Algunas medidas básicas:**

* Demorar la respuesta ante claves  erróneas;  aumentar la demora cada vez. Alertar si hay demasiados intentos.
* Registrar todas las entradas. Cada vez que un usuario entra, chequear cuándo y  desde  dónde entró la vez anterior.
* Hacer chequeos periódicos de claves fáciles de adivinar, procesos que llevan demasiado tiempo corriendo, permisos erróneos, actividades extrañas (por ejemplo cuando usuario está de vacaciones).

**-Control de acceso:** La seguridad de las Bases de Datos se concreta mediante mecanismos, tanto "hardware" como "software".

- El primero se denomina identificación, que procede a identificar a los sujetos (procesos, normalmente transacciones que actúan en su nombre o usuarios) que pretenden acceder a la base de datos.

- El siguiente mecanismo que actúa es el de autenticación. El proceso usual es mediante contraseñas, constituidas por un conjunto de caracteres alfanuméricos y especiales que sólo el sujeto conoce. También se puede realizar mediante algún dispositivo en poder del mismo.

- En caso de que el sujeto sea positivamente identificado y autenticado, se debe controlar el acceso que pretende a los objetos (datos y recursos accedidos por los sujetos. Por ejemplo, si se considera un SGBD relacional los recursos que deben protegerse son las relaciones, vistas y atributos). El mecanismo involucrado se denomina de control de accesos y se encarga de denegar o conceder dichos accesos en base a unas reglas, que establecen en qué condiciones el sujeto puede acceder y realizar ciertas operaciones sobre el objeto especificado. Estas reglas son dictadas por una persona con autoridad suficiente, que normalmente es el propietario de los datos o, en el caso de una organización, el administrador de la base de datos, de acuerdo con unas políticas de seguridad.

Los sujetos se pueden clasificar en las siguientes categorías:

* Usuarios, es decir, individuos simples conectados al sistema. A veces sería más útil especificar los criterios de acceso basándose en sus calificaciones y características, más que en la identidad del usuario.
* Grupos, es decir, conjuntos de usuarios.

Los mismos van a tener un conjunto de roles, conjuntos de privilegios necesarios para realizar actividades específicas dentro del sistema y procesos, que ejecutan programas en nombre de los usuarios.

Necesitan recursos del sistema para llevar a cabo sus actividades, y normalmente tienen acceso sólo a los recursos necesarios para que se puedan realizar las tareas del proceso. Esto limita el posible daño derivado de fallos del mecanismo de protección. Los privilegios de autorización establecen los tipos de operaciones que un sujeto puede ejercer sobre los objetos del sistema.

El conjunto de privilegios depende de los recursos a proteger. Por ejemplo, los privilegios típicos de un SGBD relacional son seleccionar, insertar, actualizar y eliminar. Normalmente, los privilegios están organizados jerárquicamente. Si la transacción invocada trata de modificar el contenido de la base de datos, los cambios propuestos son chequeados por el sistema de gestión de la misma, para garantizar su integridad semántica o elemental.

Así mismo, el sistema de gestión se responsabiliza de evitar accesos concurrentes a dicha base. Finaliza la transacción, con éxito o no, el citado sistema de gestión graba en un registro de auditoria todas las características de aquella. Este registro también contiene la información pertinente para la recuperación de la base de datos, caso de un fallo de ésta o una caída del sistema. Aunque este mecanismo no impide los accesos no autorizados, tiene efectos disuasorios sobre potenciales atacantes, permitiendo además encontrar puntos débiles en los mecanismos de seguridad.

-**Diferentes tipos de autorización.**

* Autorización explícita: Consiste en almacenar que sujetos pueden acceder a ciertos objetos con determinados privilegios. Se usa una Matriz de Accesos
* Autorización implícita: Consiste que una autorización definida sobre un objeto puede deducirse a partir de otras.

**5.2 Matriz de Autorización:**

La seguridad se logra si se cuenta con un mecanismo que limite a los usuarios a su vista o vistas personales. La norma es que la base de datos relacionales cuente con dos niveles de seguridad:

* Relación. Puede permitir o impedir que el usuario tenga acceso directo a una relación.
* Vista. Puede permitir o impedir que el usuario tenga acceso a la información que aparece en una vista.

Aunque es imposible impedir que un usuario tenga acceso directo a una información puede permitírsele acceso a una parte de esta relación por medio de una vista. De tal manera que es posible utilizar una combinación de seguridad al nivel relacional y al nivel de vistas para limitar el acceso del usuario exclusivamente a los datos que necesita.

Un usuario puede tener varias formas de autorización sobre partes de la base de datos. Entre ellas se encuentran las siguientes:

* Autorización de lectura, que permite leer, pero no modificar la base de datos.
* Autorización de inserción, permite insertar datos nuevos pero no modificar lo ya existente.
* Autorización de actualización, que permite insertar, modificar la información pero no permite la eliminación de datos.
* Autorización de borrado, que permite la eliminación de datos.

Un usuario puede tener asignados todos, ninguno o una combinación de los tipos de autorización anteriores. Además de las formas de autorización de acceso de datos antes mencionados, es posible autorizar al usuario para que modifique el esquema de la base de datos.

* Autorización de índice, que permite la creación y eliminación de índices.
* Autorización de recursos, que permite la creación de relaciones nuevas.
* Autorización de alteración, que permite agregar o eliminar atributos de una relación.
* Autorización de eliminación, que permite eliminar relaciones.

Las autorizaciones de eliminación y borrado difieren en cuanto a que la autorización de borrado solo permite la eliminación de tuplas. La habilidad para crear nuevas relaciones viene regulada por la autorización de recursos de tal forma que la utilización del espacio del almacenamiento puede ser controlada. La autorización de índice puede aparecer innecesariamente puesto que la creación o eliminación de un índice no altera los datos en las relaciones. Más bien los índices son una estructura para realizar mejoras.

La forma fundamental de autoridad es la que se le da al administrador de la base de datos. El administrador de la base de datos puede entre otras cosas autorizar nuevos usuarios, reestructurar la base de datos, etc.

**6-Otras amenazas y ataques posibles:**

* **Virus:** Un virus es parecido a un gusano, en cuanto  se reproduce, pero la diferencia es que no es un programa  por  sí  sólo, sino  que  es  un  trozo de código que se adhiere a un programa legítimo, contaminándolo. Cuando un programa  contaminado se ejecuta, ejecutará también el código del virus, lo que permite nuevas  reproducciones,  además  de alguna acción (desde un simple mensaje inocuo hasta la destrucción de todos los archivos).
* **Caballo de troya:** Un caballo de  troya es un programa aparentemente útil que contiene un trozo de código que hace algo no deseado.
* **Puerta trasera:** Una puerta trasera es un punto de entrada secreto, dejado por los implementadores del sistema para saltarse los procedimientos normales de seguridad.
* **Caza claves:** Dejar corriendo en un terminal un programa que pida "login:" y luego "password:", para engañar a los usuarios de modo que estos revelen su clave.
* **Soborno:** Sobornar al administrador para que suelte la clave.

A continuación se hace una descripción de algunas de las posibles soluciones para las amenazas mencionadas:

**Cifrado de Datos:** Su objetivo es el de hacer no entrantes los datos a usuarios no autorizados que sean capaces de acceder a ellos.

Existen numerosos métodos de cifrado de datos o métodos criptográficos. Los sistemas de cifrado más antiguos y más conocidos son los que hacen uso de una clave privada (normalmente un número) para, mediante un conjunto de transformaciones matemáticas, mantener oculto el significado de determinados datos.

**Los métodos de cifrado de datos pueden ser:**

* **Simétricos:**cuando la clave de cifrado es la misma que la de descifrado.
* **Asimétricos:**cuando ambas claves son distintas.

**Los mejores métodos de cifrado actuales**resultan prácticamente invulnerables. Algunos de los más utilizados son los siguientes:

* **Cifrado endeble.**Válidos para unos niveles de seguridad medios, pero vulnerables ante potentes sistemas de descifrado. Las claves utilizadas permiten un número de combinaciones de 240.
* **Sistemas correctos de gestión de claves.**Requieren que el usuario divida en varias partes la clave de descifrado y haga entrega de esas claves a personas de su confianza. La clave sólo puede reconstruirse en caso de que los depositarios de todas las partes se pongan de acuerdo.
* **Criptografía de clave pública.**Son criptosistemas con claves asimétricas. Estos criptosistemas se basan en que cada uno de los operadores tiene dos claves, una privada que sólo él conoce y una pública, que conocen o pueden conocer todos los intervinientes en el tráfico. Cuando el operador a quiere emitir un mensaje aplica al mismo la clave pública de B y el mensaje así cifrado se emite a B, que al recibir el mensaje le aplica su clave privada para obtener el mensaje descifrado. Por ello se denomina criptosistema asimétrico, ya que para cifrar y descifrar los mensajes se utilizan dos claves distintas.

**Los dispositivos de control de accesos se basan en una combinación de:**

* Mecanismos de identificación sobre la persona o sistema remoto.
* Mecanismos de autorización, determinando la autoridad de la persona o sistema remoto para ejecutar cada tipo de acción posible.
* Componentes aleatorios, que proporcionan protección contra el robo de claves o la suplantación de personalidad.
* Técnicas de cifrado para protegerse de la modificación de datos, de la copia, etc.
* Las técnicas avanzadas de identificación pueden utilizar características biométricas tales como el reconocimiento de la voz, medidas de retina ocular, huellas digitales, rasgos faciales, etc.

**Diccionario de Seguridad.**El diccionario de seguridad es una base de datos donde se almacenan los privilegios de acceso de los usuarios a uno o varios sistemas de información. La gestión y los procesos de esta base de datos deben ser sólo accesibles por el administrador de seguridad, siendo recomendable que esta información esté altamente protegida, incluso archivada de forma cifrada.

**Las principales entidades de esta base de datos son las siguientes:**

* Usuario.
* Perfil de usuario.
* Grupos de usuarios al que pertenece.
* Palabras clave (ocultas para el administrador).
* Sistemas a los que tiene acceso:
* Funciones, pantallas.
* Informes.
* Ficheros.
* Tipo de acceso (lectura, escritura, etc.).

**Antivirus:** Un antivirus es una aplicación informática que tiene como objetivo detectar e inutilizar los virus. Existe un tipo de equipo lógico "malicioso" orientado al sabotaje de instalaciones informáticas. Este equipo lógico lo componen segmentos de código que alteran permanente o momentáneamente el funcionamiento del equipo donde se introducen y que se "reproducen" aprovechándose de éste, o la red donde se encuentre conectado.El objetivo principal de los virus es el sistema operativo de los equipos físicos, además de los programas de aplicación y las bases de datos.Variantes modernas de los virus pueden ser los denominados polimórficos, que modifican su patrón binario cada vez que infectan un nuevo fichero, los solapados, etc.

**Productos de Seguridad de Redes de Área Local**: Las redes de área local tienen problemas de seguridad particulares y especialmente complejos, basados en su propia filosofía de permitir la libre comunicación entre los usuarios que están conectados a dicha red. Los productos de seguridad de red deben incorporar las siguientes funcionalidades:

* Facilidades de auditoría de accesos a la red.
* Controles para disparar procedimientos de recuperación y reinicio de red.
* Facilidades de copias de seguridad remotas.
* Protección de mensajes.
* Seguimiento de la ejecución de procesos de usuario.
* Informes estadísticos de tráfico, operación y fallos.
* Estos productos de seguridad de red están a menudo integrados con el sistema operativo de la red.

**7-Inyeccion SQL:**

Ejemplo de inyección SQL

SELECT \* FROM usuarios WHERE usuario = ‘ " + Usuario + “ ‘ and password =‘ "+ pass + “ ‘;

Un usuario cualquiera colocaría su nombre y su password de la siguiente manera:

SELECT \* FROM usuarios WHERE usuario = ' pepe ' and password =' 020304 '

Hasta aquí todo normal, pero un usuario podría modificar el campo password:

SELECT \* FROM usuarios WHERE usuario = ' pepe ' and password =' 020304 ' OR password LIKE '%'

Como hemos visto la inyección SQL se ha hecho con el fin de burlar la restricción de acceso, pero se pueden realizar cosas más desastrosas en la BD, como por ejemplo: **DROP TABLE usuarios**

Protegerse de una inyección SQL:

• **Asignación de mínimos privilegios**: Debe tener los privilegios necesarios, ni más ni menos.

• **Validar todas las entradas:** Especifique el tipo de dato de entrada, si son números, asegúrese de que son solo números.

• **Empleo de Procedimientos Almacenados:** Utilizar procedimientos almacenados y aceptar los datos del usuario como parámetros en lugar de comandos SQL.

• **Utilizar comillas dobles en lugar de simples:** Puesto que las comillas simples finalizan las expresiones SQL, y posibilitan la entrada de expresiones de más potencia.

La inyección SQL es fácil de evitar en la mayoría de los lenguajes de programación que desarrollan aplicaciones web. Aquí algunos ejemplos:

**En PHP:** Para MySQL la función a usar es **mysql\_real\_escape\_string**:

Ejemplo: $query\_result = mysql\_query("SELECT \* FROM usuarios WHERE nombre = \"" . **mysql\_real\_escape\_string**($nombre\_usuario) . "\"");

**En JAVA:** tenemos que usar la clase **PreparedStatement.**

En vez de: Connection con = (acquire Connection) Statement stmt = con.createStatement(); ResultSet rset = stmt.executeQuery("SELECT \* FROM usuarios WHERE nombre = '" + nombreUsuario + "';");

Habría que poner: Connection con = (acquire Connection) **PreparedStatement** pstmt = con.prepareStatement("SELECT \* FROM usuarios WHERE nombre = ?"); pstmt.setString(1, nombreUsuario); ResultSet rset = pstmt.executeQuery();

**En C#:** El siguiente ejemplo muestra cómo prevenir los ataques de inyección de código usando el objeto SqlCommand

En vez de: using( SqlConnection con = (acquire connection) ) { con. Open(); using( SqlCommand cmd = new SqlCommand("SELECT \* FROM usuarios WHERE nombre = '" + nombreUsuario + "'", con) ) { using( SqlDataReader rdr = cmd.ExecuteReader() ){ ... } } }

Habría que usar: using( SqlConnection con = (acquire connection) ) { con. Open(); using( SqlCommand cmd = new SqlCommand("SELECT \* FROM usuarios WHERE nombre = @nombreUsuario", con) ) { cmd.Parameters.AddWithValue("@nombreUsuario", nombreUsuario); using( SqlDataReader rdr = cmd.ExecuteReader() ){ ... } } }

**1.3-Hibernate:**

**1-Introduccion:** En la mayoría de las aplicaciones empresariales, si no es que en todas, una parte muy importante es el almacenamiento de datos de forma que estos datos puedan sobrevivir más allá del tiempo que nuestra aplicación está encendida (ya sea en una aplicación standalone o en una aplicación web).El almacén de datos más común son las bases de datos relacionales. Estas bases de datos no son tan fáciles utilizarlas en el almacenamiento de datos en aplicaciones orientadas a objetos (por las diferencias entre el modelo de datos de objetos y el modelo de datos relacionales), ya que para guardar un objeto debemos extraer cada una de sus propiedades que queremos persistir y armar con ellos una sentencia **INSERT** de **SQL**. De la misma forma, cuando queremos recuperar los datos de un objeto, debemos usar una sentencia **SELECT** de **SQL** y después extraer el valor de cada una de las columnas recuperadas y llenar así nuestro objeto.Esto puede no parecer un problema en aplicaciones pequeñas, pero cuando comenzamos a tener muchos objetos que debemos guardar se vuelve algo muy pesado, además de que consume mucho del tiempo del desarrollador que podría dedicar por ejemplo a mejorar la lógica de la aplicación o a realizar pruebas de la misma.

Esto se hizo de esta manera durante muchos años, hasta que comenzaron a surgir las soluciones de **Mapeo Objeto/Relacional** (**ORM** por sus siglas en inglés). El mapeo objeto/relacional se refiere a una técnica de mapear representaciones de datos de un modelo de objetos a un modelo de datos relacionales con un esquema basado en **SQL**.

**2-Que es Hibernate?:** como la definen sus autores, es una **herramienta** de mapeo objeto/relacional para Java. Además no solo se encarga del mapeo de clases Java a tablas de la base de datos (y de regreso), sino que también maneja las **queries** y recuperación de datos, lo que puede reducir de forma significativa el tiempo de desarrollo que de otra forma gastaríamos manejando los datos de forma manual con **SQL** y **JDBC**, encargándose de esta forma de las tareas comunes relacionadas con la persistencia de datos, manejando todos los problemas relacionados con la base de datos con la que estemos trabajando, de forma entendible y transparente para los desarrolladores. Entonces, si se cambia el manejador de base de datos no será necesario que se modifique todo el **SQL** que ya teníamos para adaptarse al **SQL** que maneja la nueva base de datos. Solo será necesario modificar una línea en un archivo de configuración de Hibernate, y este se encargará del resto.

**3-Otras características:** Hibernate ofrece también un lenguaje de consulta de datos llamado **HQL** (*Hibernate Query Language*), al mismo tiempo que una [API](http://es.wikipedia.org/wiki/Application_Programming_Interface) para construir las consultas programáticamente (conocida como *"criteria"*).

Hibernate para Java puede ser utilizado en aplicaciones Java independientes o en aplicaciones [Java EE](http://es.wikipedia.org/wiki/Java_EE), mediante el componente **Hibernate Annotations** que implementa el estándar [JPA](http://es.wikipedia.org/wiki/Java_Persistence_API), que es parte de esta plataforma.

**1.4-Testing**

**1. Introducción:** [Cem Kaner (Experto en pruebas de Software)](http://www.kaner.com/)define el testing como una investigación técnica de un producto bajo prueba con el fin de brindar información relativa a la calidad del software, a los diferentes actores involucrados en un proyecto.A partir de la información obtenida del testing se pueden tomar decisiones. Las decisiones pueden ser desde cuándo liberar un producto a producción, conociendo los riesgos que esto implica, hasta cómo mejorar las diferentes áreas dentro de la empresa. En definitiva el testing es un agente de cambio, lo importante es interpretar la información obtenida para que todos los actores puedan actuar en forma oportuna donde sea necesario.

Las pruebas son básicamente un conjunto de actividades dentro del desarrollo de [software](https://es.wikipedia.org/wiki/Software). Dependiendo del tipo de pruebas, estas actividades podrán ser implementadas en cualquier momento de dicho proceso de desarrollo. Existen distintos modelos de desarrollo de software, así como modelos de pruebas.

**2. Tipos de Pruebas:**

En primer lugar tenemos las **Pruebas Funcionales**. Típicamente encontraremos el comportamiento del sistema, subsistema o componente software descrito en especificaciones de requisitos o casos de uso. Es decir, con las funciones establecemos ***“*lo que el sistema hace***”.*

Estas pruebas **se definen a partir de funciones o características** (como decimos, bien descritas en documentos o bien interpretadas por los probadores) y su operabilidad con sistemas específicos, **pudiendo ejecutarse en todos los**[niveles de pruebas](http://blog.panel.es/index.php/software-qa-software-testing-levels-onion-model/)(componentes, integración, sistema, etc).

Se consideran **Pruebas de Caja Negra** (“black-box testing”) puesto que valoramos el comportamiento externo del sistema. Las **Pruebas de Seguridad** o las **Pruebas de Interoperabilidad** entre sistemas o componentes son casos especializados de las pruebas funcionales.

En segundo lugar figuran las **Pruebas no Funcionales** que incluyen las pruebas de: **Rendimiento, Carga, Estrés, Usabilidad, Mantenibilidad, Fiabilidad o Portabilidad**, entre otras. Por tanto se centran en características del software que establecen ***“*cómo trabaja el sistema*“.***

Estas pruebas también **pueden ejecutarse en todos los**[niveles de pruebas](http://blog.panel.es/index.php/software-qa-software-testing-levels-onion-model/). Las características no funcionales del software se pueden medir de diversas maneras, por ejemplo, por medio de tiempos de respuesta en el caso de pruebas de rendimiento o por número máximo de sesiones en pruebas de estrés.

Puesto que las Pruebas no Funcionales normalmente consideran el comportamiento externo del sistema, en la mayoría de los casos se utilizan técnicas de Pruebas de **Caja Negra.**

En tercer lugar, tenemos las Pruebas Estructurales. Nuevamente pueden ejecutarse en todos los niveles de pruebas y van muy bien si hemos utilizado técnicas de especificación de la estructura o arquitectura del Software. Es posible aplicar técnicas estáticas de análisis de código.

Para expresar el alcance con un conjunto de pruebas (“test suite”) que ha cubierto la estructura o arquitectura en cuestión, se utiliza el concepto de **Cobertura** (“Coverage”), normalmente en forma de porcentaje.

Es especialmente habitual utilizar **herramientas de apoyo para calcular la cobertura del código** en el caso de Pruebas de Componentes o en Pruebas de Integración de Componentes (por ejemplo, trazando la jerarquía de llamadas entre elementos). Puesto que indagamos en el comportamiento interno, estas pruebas se denominan también **Pruebas de Caja Blanca** (“white-box testing”).

Finalmente, el cuarto tipo de pruebas que nos presenta el [ISTQB](http://www.istqb.org/) son las pruebas derivadas de la realización de cambios: las **Pruebas de Regresión y las Re-pruebas**.

Una vez que un defecto ha sido corregido, toca volver a probar el software para confirmar que el defecto ha sido eliminado. Son pruebas repetidas o Re-Pruebas.

Las Pruebas de Regresión consisten en volver a probar un componente, tras haber sido modificado, para descubrir cualquier defecto introducido, o no cubierto previamente, como consecuencia de los cambios. Los defectos pueden encontrarse tanto en el software que se ha cambiado como en algún otro componente. Se ejecutan cuando se cambia el software o su entorno. El criterio para decidir la extensión de estas Pruebas de Regresión está basado en el **riesgo de no encontrar defectos en el software que anteriormente estaba funcionando correctamente.**

Las Pruebas de Regresión se realizan sobre un componente ya probado, para verificar que no presenta nuevos defectos cuando se realiza una modificación después de dichas pruebas.

Este tipo de pruebas deben ser **repetibles** si han de usarse para pruebas de confirmación (o aseguramiento) y regresión. Los conjuntos de pruebas de regresión (“Regression test suites“) suelen ser bastante estables por lo que son muy buenos candidatos para **actividades de automatización de pruebas.**

**1.5-Bibliografia:**

[**https://es.wikipedia.org/wiki/Android**](https://es.wikipedia.org/wiki/Android)

[**https://sites.google.com/site/swcuc3m/home/android**](https://sites.google.com/site/swcuc3m/home/android)

[**http://ingenieriacognitiva.com/developer/cursos/AndroidBasico/chapters/c2.php**](http://ingenieriacognitiva.com/developer/cursos/AndroidBasico/chapters/c2.php)

[**http://es.slideshare.net/Drakonis11/integridad-y-seguridad-en-las-bases-de-datos-presentation**](http://es.slideshare.net/Drakonis11/integridad-y-seguridad-en-las-bases-de-datos-presentation)

[**http://bellaleislyeva.blogspot.com.uy/2011/05/seguridad-y-encriptacion-de-bases-de.html**](http://bellaleislyeva.blogspot.com.uy/2011/05/seguridad-y-encriptacion-de-bases-de.html)

[**https://es.wikipedia.org/wiki/Hibernate**](https://es.wikipedia.org/wiki/Hibernate)

[**http://www.javatutoriales.com/2009/05/hibernate-parte-1-persistiendo-objetos.html**](http://www.javatutoriales.com/2009/05/hibernate-parte-1-persistiendo-objetos.html)

[**http://migranitodejava.blogspot.com.uy/2011/08/introduccion-hibernate.html**](http://migranitodejava.blogspot.com.uy/2011/08/introduccion-hibernate.html)

[**http://www.ces.com.uy/**](http://www.ces.com.uy/)

[**https://es.wikipedia.org/wiki/Pruebas\_de\_software**](https://es.wikipedia.org/wiki/Pruebas_de_software)

[**http://blog.panel.es/index.php/software-qa-cuales-son-los-tipos-de-pruebas-software/**](http://blog.panel.es/index.php/software-qa-cuales-son-los-tipos-de-pruebas-software/)