**FORMATO PARA EL DESARROLLO DE COMPONENTE FORMATIVO**

|  |  |
| --- | --- |
| PROGRAMA DE FORMACIÓN | ADSO |

|  |  |  |  |
| --- | --- | --- | --- |
| COMPETENCIA | 220501096. Desarrollar la solución de *software* de acuerdo con el diseño y metodologías de desarrollo. | RESULTADOS DE APRENDIZAJE | 20501096-04. Codificar el *software* de acuerdo con el diseño establecido. |

|  |  |
| --- | --- |
| NÚMERO DEL COMPONENTE FORMATIVO | 24 |
| NOMBRE DEL COMPONENTE FORMATIVO | Construir aplicativo según las características de la arquitectura Android |
| BREVE DESCRIPCIÓN | En este componente formativo se abordan los conceptos claves para el diseño y desarrollo de aplicaciones móviles nativas, junto con sus elementos, vistas, maquetación, entornos de desarrollo, lenguajes de programación. |
| PALABRAS CLAVE | SDK, Android Studio, *Manifest, Layout, Firebase.* |

|  |  |
| --- | --- |
| ÁREA OCUPACIONAL | 2 - CIENCIAS NATURALES, APLICADAS Y RELACIONADAS |
| IDIOMA | Español |

1. **TABLA DE CONTENIDOS:** 
   * + 1. Estructura de un proyecto en Android Studio
       2. Interfaz de usuario en Android

2.1. *Layouts*

2.2. Controles básicos

3. Eventos *listeners*

*4.* Navegación (*intents)*

5. Tareas asincrónicas

6. Persistencia de datos

6.1. Bases de datos local SQLite

6.2. Bases de datos en tiempo real

7. Multimedia

*7.1. MediaPlayer*

*7.2. VideoView*

1. **INTRODUCCIÓN**

Actualmente, los sistemas operativos para dispositivos móviles permiten la construcción de aplicaciones mediante el uso de un *software* llamado SDK. Tanto Android como iOS ofrecen sus respectivas herramientas para crear aplicaciones nativas. Las características de los dispositivos móviles, como GPS, cámara y navegación por internet, se aprovechan en el desarrollo de aplicaciones, permitiendo ideas interesantes y útiles para los usuarios.

Este componente formativo se enfoca en el desarrollo de aplicaciones nativas para la plataforma Android, utilizando su entorno de desarrollo oficial, Android Studio, y el lenguaje Java.

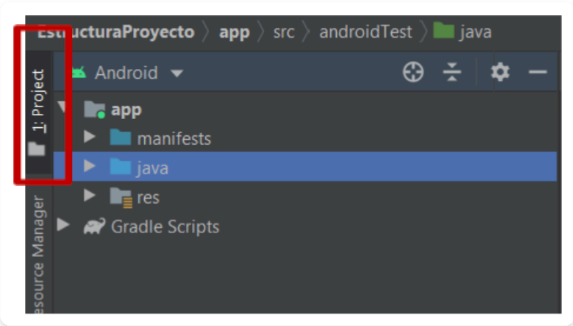
1. **DESARROLLO DE CONTENIDOS:**
2. **Estructura de un proyecto en Android Studio**

Las actualizaciones en Android Studio avanzan rápidamente. Para este componente, los ejemplos se presentan para la versión 4.1.3.

En Android Studio, un proyecto contiene todos los elementos necesarios para el desarrollo de una aplicación: desde código fuente y recursos hasta código de prueba y configuraciones de compilación. Al iniciar un nuevo proyecto, Android Studio crea la estructura necesaria para todos los archivos y los organiza en la ventana Project, ubicada en el lado izquierdo del IDE **(accede a ella mediante View > Tool Windows > Project),** o utilizando el acceso directo Alt + 1.

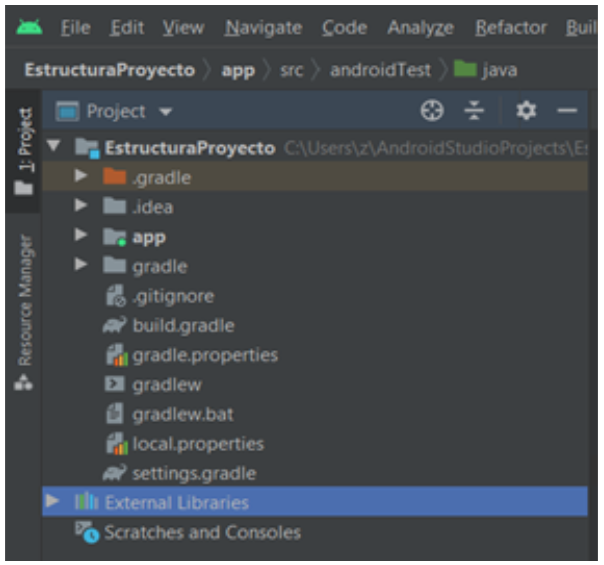
A continuación, se presenta la estructura:

1. Imagen estructura de proyecto en Android Studio



Ahora bien, las vistas de proyectos en Android, o la visualización completa de la estructura de archivos del proyecto, se realizan mediante la opción "Project" del menú desplegable en la parte superior de la ventana, como se ilustra a continuación.

1. Vista estructura del proyecto



Los proyectos de Android Studio tienen unos componentes principales que se explican en el siguiente video, los cuales se distribuyen entre:

* Módulos
* Archivos de proyecto

A green robot with blue eyes

Description automatically generated

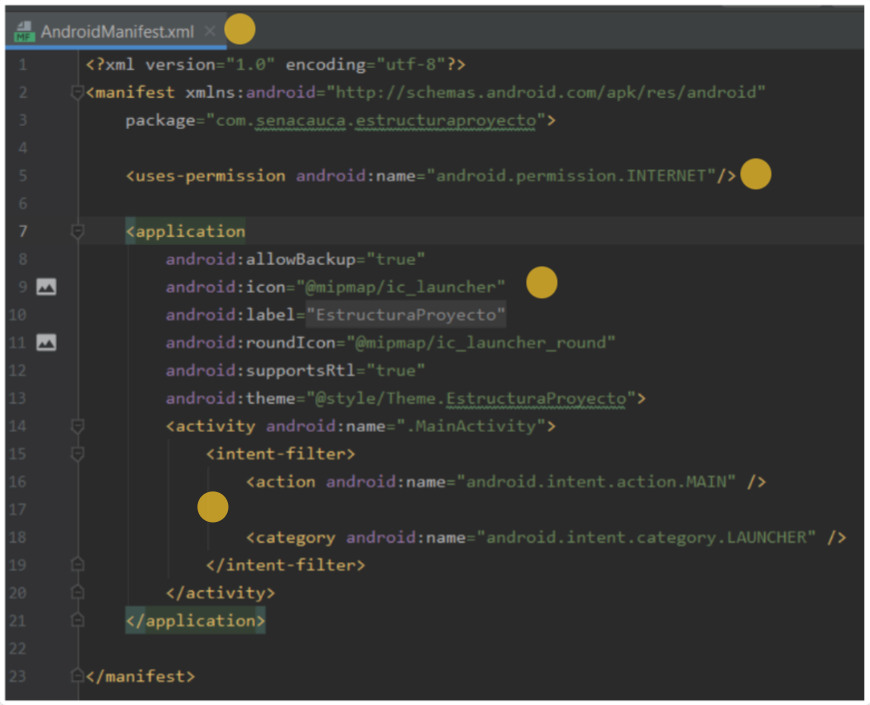
<https://www.youtube.com/watch?v=-OauYZEnlvE>

***Manifest***

En el *manifest* se encuentra la información esencial de una aplicación para las herramientas de desarrollo de Android, el sistema operativo Android y Google Play. El *manifest* es un archivo en lenguaje XML donde se definen aspectos principales de la aplicación, como su identificación (nombre, ícono, estilos), sus componentes (pantallas, servicios) y los permisos necesarios para su ejecución.

A continuación, se presentan algunos de los elementos más importantes que se deben declarar en el archivo *manifest:*

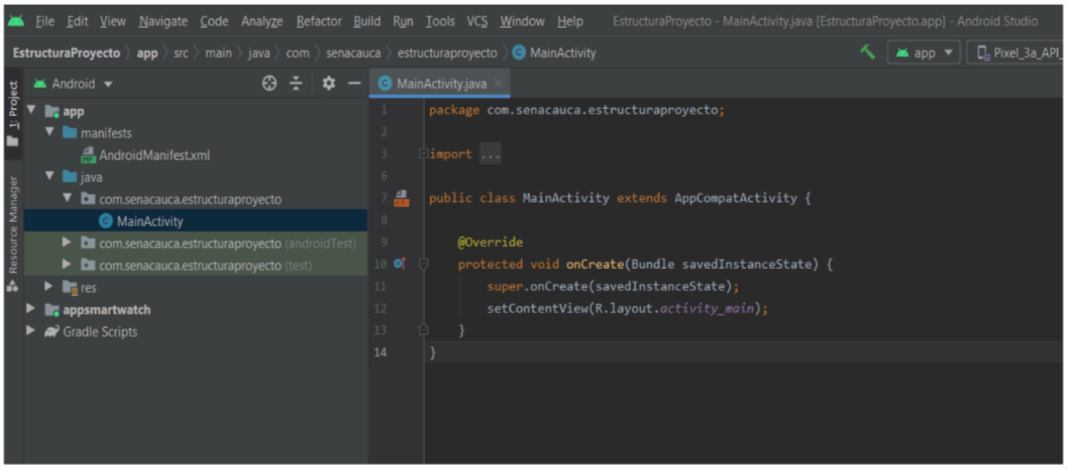
1. Archivo manifest



**Java**

La carpeta java contiene todo el código fuente de la aplicación, organizado en paquetes, clases, interfaces, POJOs (P*lain Old Java Objects*), entre otros. Inicialmente, Android Studio crea el código básico de la pantalla principal de la aplicación (actividad o *activity*), que por defecto se llamará *MainActivity,* siempre bajo la estructura del paquete java definido durante la creación del proyecto.

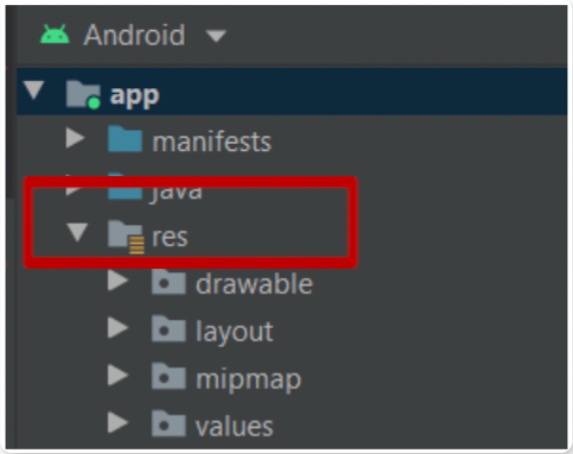
1. Clase MainActivity de la carpeta java



***Res***

En esta carpeta se encuentran todos los recursos del proyecto que no contienen código, como diseños XML, cadenas de texto de la interfaz de usuario y también imágenes de mapa de *bits,* organizados en subdirectorios.

1. Imagen estructura carpeta res



A continuación, se explican las subcarpetas:

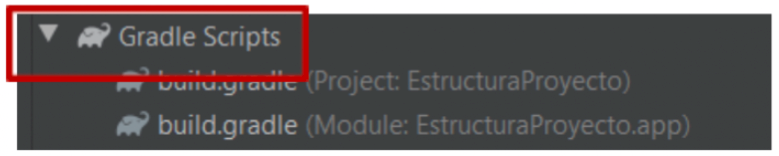
|  |
| --- |
| Slide  CF024\_1\_Estructura de un proyecto en Android Studio |

***Gradle***

*Gradle* permite la compilación del proyecto. En este, es posible configurar la versión del SDK que Android utilizará para compilar, la versión mínima de Android que soportará la aplicación, referencias a las bibliotecas externas utilizadas, entre otros aspectos.

En un proyecto pueden existir varios archivos *build.gradle*, para definir parámetros en distintos niveles. Por ejemplo, puede haber un *build.gradle* a nivel de proyecto y otro a nivel de módulo dentro de la carpeta */app*. El primero definirá parámetros globales para todos los módulos del proyecto, mientras que el segundo solo afectará a cada módulo en particular.

1. Gradle Scripts

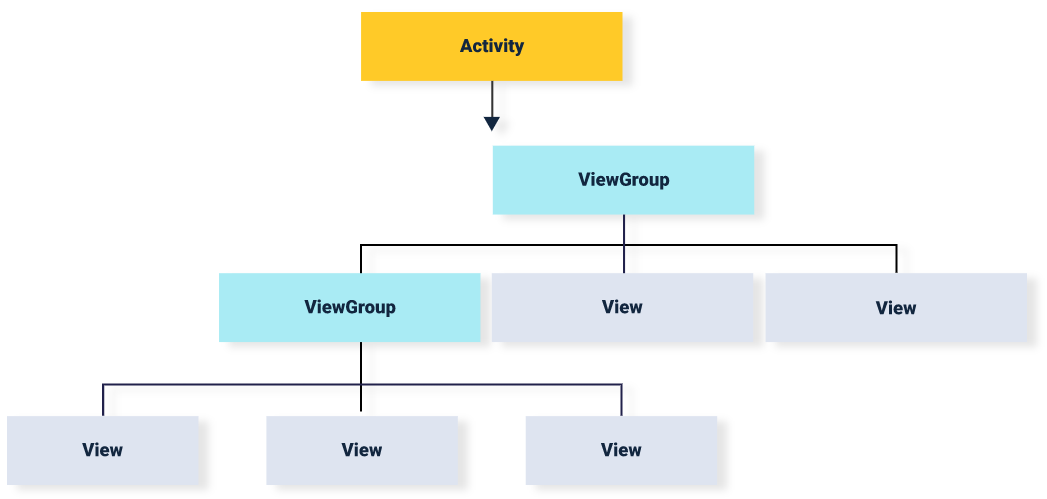


1. **Interfaz de usuario en Android**

La interfaz de usuario se define a través de la estructura de un diseño para la aplicación, como en el caso de una actividad en Android. La creación de los elementos del diseño se realiza utilizando una jerarquía de objetos *View* y *ViewGroup*. Los *View* son los elementos que el usuario puede interactuar con, como botones, textos e imágenes, mientras que un *ViewGroup* es un contenedor no visible que define la estructura del diseño de los *View* y otros objetos.

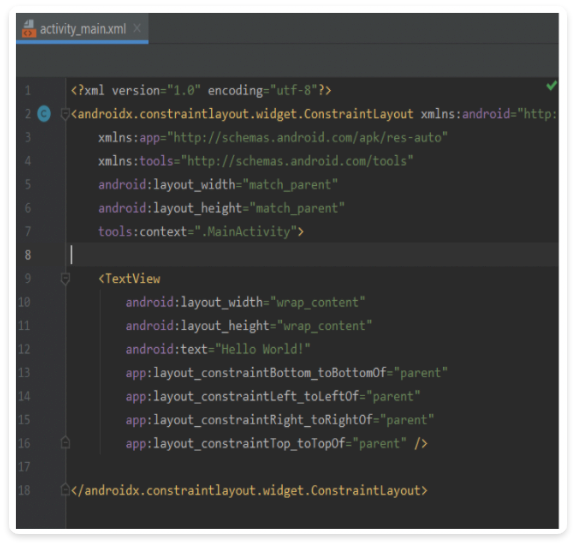
Los objetos *ViewGroup* se denominan *Layout* (diseños) y existen en varios tipos, dependiendo de la distribución y la posición de sus *View*. Ejemplos de estos son *LinearLayout*, *RelativeLayout*, entre otros.

1. Ilustración jerarquía de diseño

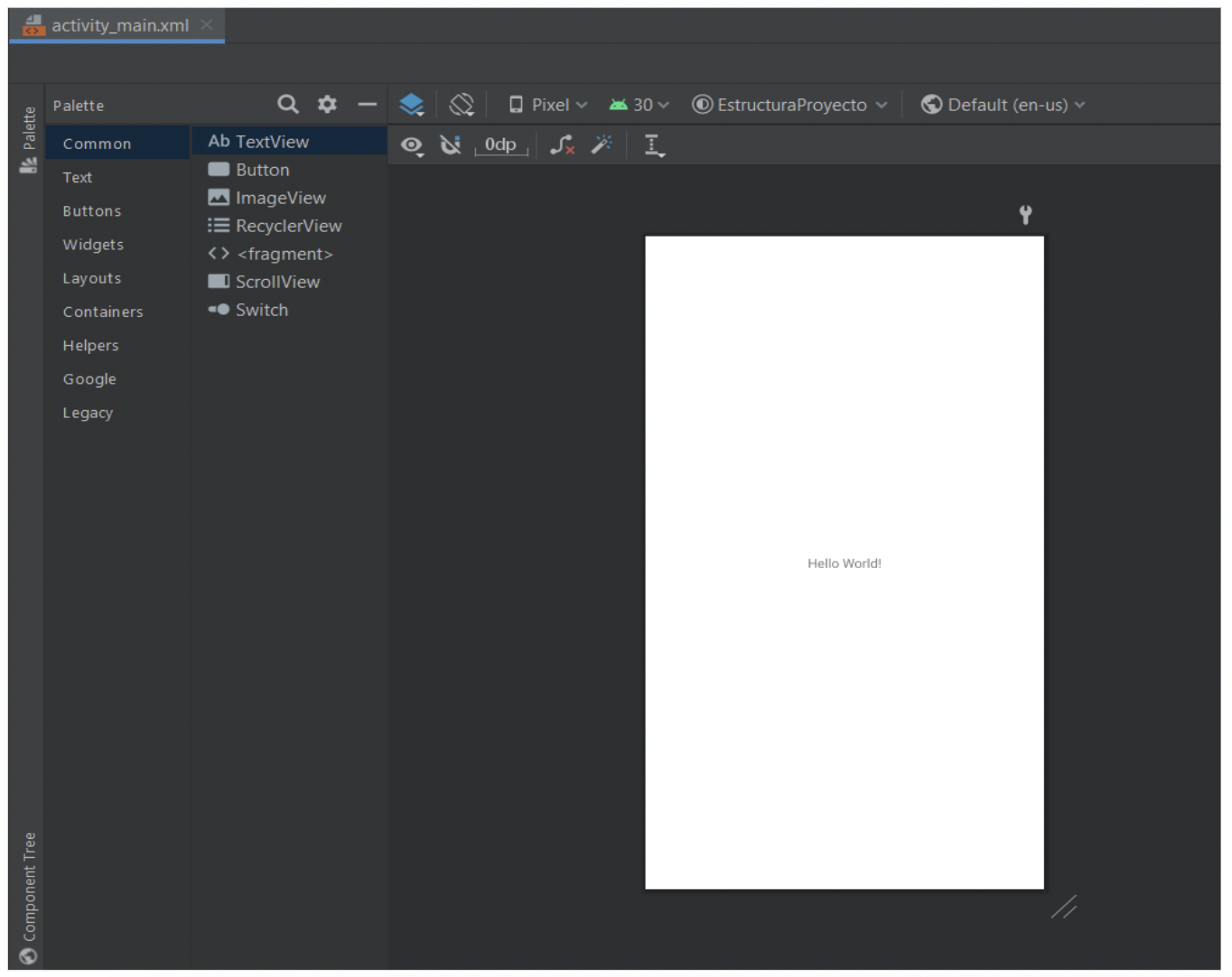


El diseño en Android se puede trabajar de dos maneras: en XML, o en el editor de diseño (mediante una interfaz de arrastrar y soltar).

1. Vista diseño XML



1. Vista diseño editor



A screenshot of a cell phone

Description automatically generated**2.1. *Layouts***

En Android Studio, los componentes visuales (como *Button, EditText, TextView, etc.*) se organizan mediante contenedores llamados *Layout*. Los *Layouts* son elementos no visuales que permiten controlar la organización, posición y dimensiones de los *View* que se crean dentro de ellos. Al igual que muchos otros componentes contenedores, son capaces de contener otros controles, es decir, se puede tener un contenedor dentro de otro contenedor.

Existen diferentes tipos de *Layout*, entre los más comunes están: *LinearLayout*, *TableLayout* y *ConstraintLayout*. A continuación, se profundiza en cada uno:

|  |
| --- |
| Acordeón  CF024\_2.1\_*Layouts* |

**2.2. Controles básicos**

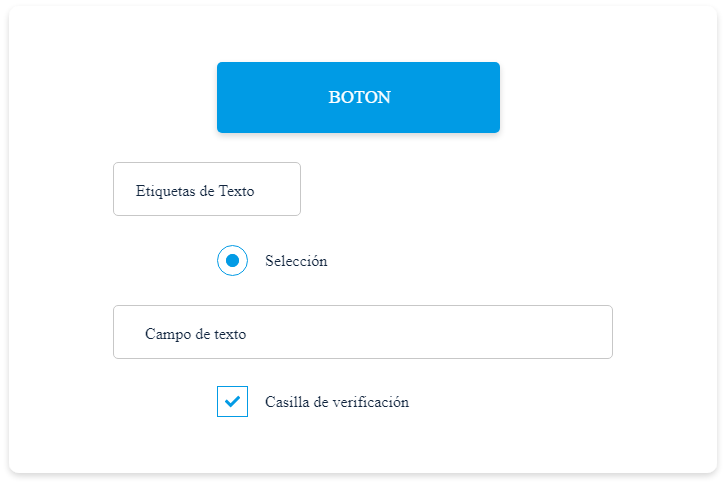
En Android Studio se encuentra una amplia variedad de controles que se pueden utilizar en la interfaz de usuario, tales como botones (*Button*), campos de texto (*EditText*), vista de imágenes (*ImageView*), casillas de verificación (*CheckBox*), botones de selección (*RadioButton*), etiquetas de texto (*TextView*), listas (*ListView)*, entre otros.

Existen tres propiedades indispensables en la creación de los controles mencionados anteriormente:

Los valores que pueden tomar las propiedades de Android: **layout\_width,** y Android: **layout\_height** los cuales aplican para todos los controles son:

Estas tres propiedades básicas son las principales en los componentes que se describen a continuación.

1. Controles básicos



**Botones**

Un botón (*Button)* es un elemento de la interfaz en el que el usuario puede tocar o hacer clic para realizar una acción. A continuación, se presenta un ejemplo de un *Button* con sus propiedades básicas:

**Tabla 1.** Propiedades básicas *Button*

| Propiedades | Definición |
| --- | --- |
| **android:layout\_width** | Define el ancho de la vista. |
| **android:layout\_height** | Define el alto de la vista. |
| **android:id** | Define el identificador asociado a la vista para ser referenciado por código java. |
| **android:text** | Define el texto asociado a la vista. |

XML Creación *Button:*

|  |
| --- |
| <Button  android:id="@+id/button"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:text="Boton" /> |

Y el botón quedaría como se presenta a continuación:

1. Vista diseño Button



**Texto e imágenes**

Uno de los controles más utilizados son los referentes a los textos e imágenes, ya que son con los que más interactúa el usuario. A continuación, se describen tres de los componentes básicos más imprescindibles para el desarrollo de nuestras aplicaciones: los textos o etiquetas (*TextView*), las imágenes (*ImageView*) y los cuadros de texto (*EditText)*.

***TextView:*** es un control utilizado para desplegar contenido textual generalmente estático. Tiene la propiedad de acoplarse al control que lo contenga y no captura el foco en ningún momento, por lo que generalmente no se programa ningún evento sobre él.

Sus propiedades básicas son las siguientes:

**Tabla 2.** Propiedades básicas *TextView*

|  |  |
| --- | --- |
| ****Propiedades**** | ****Definición**** |
| **android:layout\_width** | **Define el ancho de la vista.** |
| **android:layout\_height** | **Define el alto de la vista.** |
| **android:id** | **Define el identificador asociado a la vista para ser referenciado por código java.** |
| **android:text** | **Establece cual es el contenido para mostrar por el *TextView.***  **Existen dos formas de asociar el texto a una vista, la primera forma y más simple es asociar el texto de forma directa:**  **android:text=“Hola Mundo”**  **La segunda forma es por medio de archivo *string* XML ubicado dentro de la carpeta *values* de los recursos del proyecto y asociar cada uno de los textos con una llave respectiva.**  **Ya luego se referencia el contenido por medio de este archivo XML de la siguiente forma: android:text="@string/boton2"** |
| **android: textColor** | **Esta propiedad nos permite definir cuál va a ser el color del texto.** |
| **android: textSize** | **Con esta propiedad podemos definir el tamaño de la fuente la unidad es sp.**  **Ejemplo: android:textSize="20sp"** |
| **android: textStyle** | **Define el estilo de la fuente a usar, tiene los siguientes valores:**  android:textStyle="bold"  android:textStyle="italic"  android:textStyle="normal" |
| **android: typeface** | **Esta propiedad me permite cambiar el tipo de fuente dentro de un conjunto de 4 posibles opciones: normal, *sans, serif y monospace,* se pueden descargar otro tipo de fuentes.** |

Ejemplo de *TextView* con sus propiedades básicas:

|  |
| --- |
| <TextView  android:id="@+id/txtTexto"  android:text="@string/control\_de\_texto"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:textStyle="bold"  android:textColor="@color/purple\_500"  android:typeface="monospace"  android:textSize="40sp" /> |

El *TextView* quedaría como se presenta a continuación:

1. Vista diseño TextView

A close up of a number

Description automatically generated

**ImageView**: es un control de imágenes que permite acceder a los recursos de imágenes *(drawable o mipmap)* y publicar en la aplicación. Su propiedad principal es **android:src**, que permite indicar la imagen a exponer del proyecto. Lo primero que se debe hacer es copiar la imagen que se quiere presentar en la carpeta ***drawable*** del proyecto (/**res/drawable)** y, después, en el XML se debe indicar la ruta de origen así: **android:src="@drawable/imagen"**.

Aparte de las propiedades de *width* y *height* que se encuentran en todos los *view*, están las siguientes propiedades básicas para el *ImageView*:

**Tabla 3.** Propiedades básicas *ImageView*

|  |  |
| --- | --- |
| **Propiedades** | **Definición** |
| **android:src** | Establece la imagen del *ImageView.* |
| **android:scaleType** | Controla cómo se debe cambiar el tamaño o mover la imagen para que coincida con el tamaño de este *ImageView*. |
| **android:tint** | El color de tinte de la imagen. |

Ejemplo de *ImageView* con sus propiedades básicas:

|  |
| --- |
| <ImageView  android:id="@+id/txtTexto"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:src="@drawable/andy"  android:scaleType="centerInside" /> |

El *ImageView* quedaría como se presenta a continuación:

1. Vista diseño ImageView



***EditText****:* es un elemento de la interfaz que permite al usuario ingresar y editar texto. Una de sus propiedades más funcionales es **inputType**, que define el tipo de entrada que tendrá el cuadro de texto. El valor se aplica dependiendo de lo que se quiera ingresar: por ejemplo, un correo electrónico (***textEmailAddress***), edad *(****number***), apellido *(****text***), entre otros. La elección del tipo de entrada configura el tipo de teclado que se expondrá.

**Tabla 4.** Propiedades básicas de *EditText*

|  |  |
| --- | --- |
| **Propiedad** | **Definición** |
| **android:inputType** | Establece el tipo de dato que se ingresará en el campo de texto. Para esta propiedad existen valores como: ***text****,* ***number****,* ***textUri****,* ***textPassword****,* ***phone****,* ***date****,* entre otros. |
| **android:hint** | Permite desplegar un texto sobre el control a manera de ayuda mientras el usuario aún no ingresa información. Se prefiere sobre la propiedad ***text****.* |
| **android:text** | Al igual que en *TextView,* se puede definir cuál es el texto que se mostrará en el control |

Ejemplo de *EditText* con sus propiedades básicas:

|  |
| --- |
| <EditText  android:id="@+id/txtTexto"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:hint="Escriba su Nombre"  android:inputType="text" /> |

El *EditText q*uedaría como se presenta a continuación:

1. Vista diseño EditText

****

***Checkbox y Radiobutton***

Otro de los controles muy utilizados en las aplicaciones son los que permiten hacer una selección o un chequeo de alguna opción. Android cuenta con dos tipos: *checkboxes y radiobuttons.*

**CheckBox:** Las casillas de verificación permiten que el usuario seleccione una o más opciones de un grupo. Aparte de las propiedades de *width y height* que se encuentran en todas las vistas *(view)*, las siguientes son básicas para el *Checkbox:*

**Tabla 5.** Propiedades básicas *Checkbox*

|  |  |
| --- | --- |
| **Propiedades** | **Definición** |
| android:checked | Establece el estado actual del *checkbox*. |
| android:text | Define el texto del *checkbox*. |
| android:id | Define el identificador único asociado a la vista para ser referenciado por código Java. |

Para definir un control de este tipo, se utiliza el siguiente código, que define dos *checkboxes* con los textos "español" e "inglés":

|  |
| --- |
| <CheckBox  android:id="@+id/txtTexto"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:text="@string/espa\_ol"  android:checked="true" />  <CheckBox  android:id="@+id/txtTexto2"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:text="@string/ingles"  android:checked="false" /> |

Los *checkbox* quedarían como se presenta a continuación:

1. Vista diseño Checkbox



***RadioButton****:* a diferencia del *CheckBox,* el control *RadioButton* permite realizar una selección única entre una lista de opciones. En Android, un grupo de botones *RadioButton* se define mediante un elemento *RadioGroup,* el cual contiene todos los elementos *RadioButton* necesarios.

**Tabla 6.** Propiedades básicas de *RadioButton*

|  |  |
| --- | --- |
| **Propiedad** | **Definición** |
| **android:checked** | Establece el estado actual de selección por defecto del *RadioButton.* |
| **android:text** | Define el texto que se mostrará en el *RadioButton.* |
| **android:id** | Define el identificador único asociado a la vista para ser referenciado por código Java, al igual que en los otros controles. |

Para definir un control de este tipo, se utiliza como componente padre un *RadioGroup* y como hijos dos *RadioButton.* El *RadioGroup* es el que permite que solo sea seleccionable una sola opción; si se crean RadioButton sin un *RadioGroup,* todas las opciones serán seleccionables, lo que no es el propósito de este componente.

En las propiedades del *RadioGroup* se puede definir la orientación de los *RadioButton*, que puede ser vertical u horizontal.

El siguiente código presenta la creación de dos *RadioButton* con los textos de "femenino" y "masculino", los cuales están dentro de un *RadioGroup* con la propiedad de orientación vertical.

|  |
| --- |
| <RadioGroup  android:id="@+id/rg1"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:orientation="vertical">  <RadioButton  android:id="@+id/rb1"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:text="@string/femenino"  android:checked="true" />  <RadioButton  android:id="@+id/rb2"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:text="@string/masculino"  android:checked="false" />  </RadioGroup> |

Los *RadioButton* quedarían como se presenta a continuación:

1. Vista diseño RadioButton



**Controles de selección**

Los diferentes controles que Android ofrece para seleccionar opciones dentro de una lista de posibilidades se llaman *Spinner* (para listas desplegables) y *ListView* (para listas fijas).

***Spinner***

Este permite seleccionar un valor de un conjunto. Al hacer clic sobre él, se presenta una especie de lista emergente con todas las opciones disponibles. Su propiedad más importante es **android:entries="@array/lista"**, que permite fijar el contenido de datos que se quiere mostrar en la lista.

Para añadir una lista de este tipo, se deben seguir los pasos que se mencionan a continuación:

1. En los recursos**/res/values/strings.xml,** se debe crear un arreglo de *strings (string-array*) con los valores que se quieren presentar en el *spinner.*

|  |
| --- |
| <resources>  <string name="app\_name">RadioButton</string>  <string-array name="ciudades">  <!-- Se crea el Arreglo, en este caso se coloca el nombre de ciudades -->  <item>Popayán</item> <!-- Se crean los elementos del arreglo -->  <item>Cali</item>  <item>Bogotá</item>  <item>Cartagena</item>  <item>Medellín</item>  <item>Pasto</item>  </string-array> <!-- Cierre del arreglo -->  </resources> |

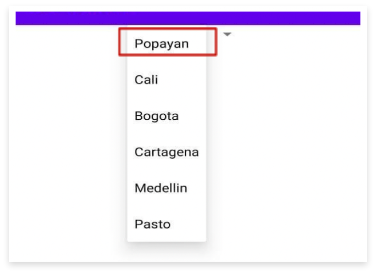
1. Ahora en el **/res/layout** se crea el siguiente código para crear el *Spinner* y enlazar el **string-array** a través de la propiedad *entries* como se presenta:

|  |
| --- |
| <Spinner  android:id="@+id/spCiudades"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:entries="@array/ciudades" /> |

El código enseña la creación de un *Spinner* con sus propiedades básicas *height*, *width* y *id*, y la propiedad *entries*, donde se relaciona el array "ciudades" creado en los recursos.

El *Spinner* se presentaría de la siguiente manera:

1. Vista Diseño Spinner ciudades



***ListView***

El *ListView* permite mostrar al usuario una lista de opciones seleccionables directamente sobre el propio control, sin listas emergentes como en el caso del control *Spinner.* Si existen más opciones de las que se pueden mostrar en el control, se podrá hacer *scroll* sobre la lista para acceder al resto de los elementos.

|  |  |
| --- | --- |
| Sección media de un hombre de negocios usando una tableta digital | Para crear el *ListView,* al igual que en el *Spinner,* también se debe crear el arreglo de elementos que se quiere mostrar en la lista. Para este ejemplo, se utilizará el mismo arreglo de ciudades que se usó para el *Spinner.* |

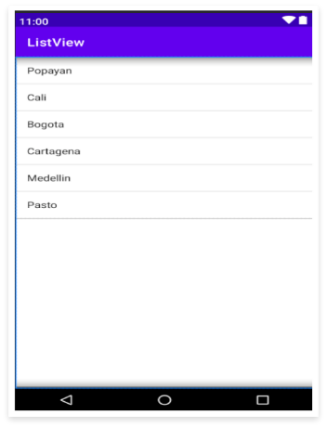
Cuando los elementos que se quieren mostrar en la lista o en el *Spinner* no son estáticos, se crea el arreglo desde el código Java.

Ejemplo de creación de un *ListView:*

|  |
| --- |
| <ListView  android:layout\_gravity="center"  android:id="@+id/listaCiudades"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:entries="@array/ciudades" /> |

El código describe la creación de un *ListView,* utilizando las mismas propiedades que el *Spinner,* y la propiedad *entries,* donde se relaciona el *array* "ciudades" creado en los recursos.

1. Lista diseño Listview ciudades



1. **Eventos *listeners***

En Android, la interacción del usuario con la aplicación se gestiona a través de eventos. Por ejemplo, cuando se toca una vista (como un botón), se llama al método **onTouchEvent()** del objeto, utilizando los objetos de escucha de eventos para detectar la interacción del usuario.

La ejecución de los eventos se realiza en el código Java ubicado en la carpeta **/java**.

Para fijar un *listener,* se debe implementar y agregar los métodos del evento correspondiente a cada *View* mediante el método **setOn**. Para realizar el ejemplo del evento *listener* de un *Button*, se deben seguir algunos pasos:

|  |
| --- |
| Acordeón  CF024\_3\_Eventos listeners |

1. **Navegación (*intents)***

La navegación entre pantallas en Android se realiza a través de los *Intents*. El *Intent* es básicamente una intención de realizar una acción; es la forma de comunicación en Android que permite la interacción entre componentes de la misma aplicación o entre diferentes aplicaciones (*Activities, Services, Broadcasts*, etc.).

Existen dos tipos de *Intents*:

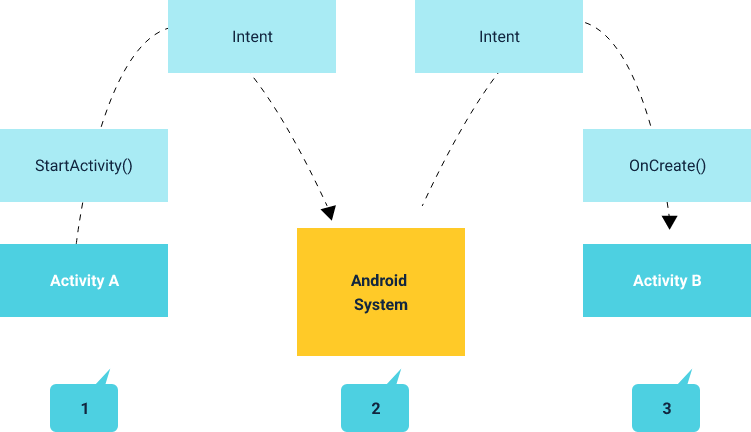
**Intent explícito**

Se utiliza para llamar a un componente específico. Por ejemplo, si se tiene una aplicación que contiene dos actividades, Actividad A y Actividad B, y se requiere navegar desde la Actividad A hasta la Actividad B, se define una intención explícita.

Para ello, se utiliza la clase *Intent* donde se especifica dónde está y a dónde se quiere ir:

|  |
| --- |
| Intent intent = new Intent(ActividadA.this, ActividadB.class);  startActivity(intent); |

1. **Uso intent explicito**



La figura presenta cómo se utiliza un *intent* para iniciar una actividad. Cuando el objeto *Intent* nombra un componente de actividad específico de forma explícita, el sistema inicia ese componente al instante.

***Intent* implícito**

A diferencia de los explícitos, los *intent* implícitos no nombran el componente específico. Se usan para iniciar un componente específico de la aplicación. Por ejemplo, si se desea abrir la cámara, una ubicación en un mapa o una página web, se utilizan los *intent* implícitos para solicitar que se activen estos servicios.

**Ejemplo:**

|  |
| --- |
| Intent intent = new Intent(Intent.ACTION\_VIEW, Uri.parse("http://www.example.com"));  startActivity(intent); |

1. **Tareas asincrónicas**

Todos los componentes de una aplicación Android, como las actividades, los servicios o los *broadcast receivers*, se ejecutan en el mismo hilo de ejecución, llamado hilo principal, *main thread* o *GUI thread*. Como sugiere este último nombre, también es el hilo donde se realizan todas las operaciones que gestionan la interfaz de usuario de la aplicación. Por ello, cualquier operación larga o costosa que se realice en este hilo bloqueará la ejecución del resto de componentes de la aplicación y, por supuesto, también la interfaz, produciendo un efecto evidente de lentitud, bloqueo o mal funcionamiento en general para el usuario.

|  |  |
| --- | --- |
| A yellow and blue arrows  Description automatically generated | **Android utiliza la clase AsyncTask para ejecutar tareas en segundo plano.** |

La forma básica de utilizar la clase **AsyncTask** consiste en crear una nueva clase que extienda de ella y sobrescribir varios de sus métodos, distribuyendo la funcionalidad de la tarea entre ellos. Estos métodos son los siguientes:

|  |
| --- |
| Tarjetas  CF024\_5\_Tareas asincrónicas |

Estos métodos tienen una particularidad esencial para nuestros intereses. El método **doInBackground()** se ejecuta en un hilo secundario, por lo tanto, no se puede interactuar con la interfaz en su interior. Sin embargo, todos los demás métodos se ejecutan en el hilo principal, lo que significa que dentro de ellos se puede hacer referencia directa a los controles de usuario para actualizar la interfaz.

**6. Persistencia de datos**

En el desarrollo móvil, es indispensable que las aplicaciones puedan guardar datos, ya sea para no perder información en una nueva ejecución o para manejar información en bases de datos. La persistencia de datos es una habilidad esencial para que un desarrollador cree aplicaciones de alta calidad. Uno de los métodos más comunes de almacenamiento en Android es el de las Bases de Datos en SQLite.

**6.1. Bases de datos local SQLite**

SQLite es un motor de bases de datos relacional de código abierto y actualmente muy popular, ya que ofrece particularidades muy útiles como su pequeño tamaño, ligereza, no necesitar servidor, requerir poca configuración y guardar toda la base de datos en un solo fichero. Su uso es muy habitual en aplicaciones de tamaño pequeño.

Para la administración de una base de datos SQLite se utiliza la clase **SQLiteDatabase**:

**SQLiteDatabase**: tiene métodos para crear, eliminar, ejecutar comandos SQL y realizar otras tareas comunes de administración de bases de datos.

Los métodos más utilizados son:

**Tabla 7.** Métodos de administración de bases de datos de la clase SQLiteDatabase

|  |  |
| --- | --- |
| **Acción** | **Método** |
| **Insertar datos** | db.insert |
| **Consultar datos** | db.rawQuery() y db.query() |
| **Eliminar datos** | db.delete |
| **Actualizar datos** | db.update |
| **Ejecutar sentencia SQL** | db.execute |

Para reflejar la utilidad de las bases de datos SQLite, se realizará un ejemplo de una base de datos para registro de usuarios y de entidades. Se tendrá una actividad de solicitud de datos para la creación, actualización y borrado en la base de datos, permitiendo gestionar la información de forma persistente.

**6.2. Bases de datos en tiempo real**

Una base de datos en tiempo real se refiere a un sistema de base de datos que procesa datos en tiempo real, tomando los datos entrantes a medida que fluyen del servidor. La base de datos en tiempo real de Firebase (*Firebase Realtime Database*) es sin duda uno de los servicios más populares de la plataforma.

Firebase es una plataforma para el desarrollo de aplicaciones web y móviles, creada por James Tamplin y Andrew Lee en 2011, y adquirida por Google en 2014. Provee servicios muy útiles que se pueden incorporar en aplicaciones Android (además de otros sistemas operativos), tales como:

|  |
| --- |
| Slide  CF024\_6.2.\_Bases de datos en tiempo real (1) |

Principales características de Firebase Realtime Database:

Para crear un proyecto con Firebase se deberán tener lo siguiente:

* A yellow folded paper

  Description automatically generatedUna cuenta de Gmail para acceder a la consola de Firebase <https://console.firebase.google.com>
* Un proyecto de Android Studio para poder hacer la conexión con la base de datos de Firebase.

**Configuración de Firebase en el proyecto de Android Studio**

Para la configuración de Firebase en el proyecto de Android Studio se enumera los pasos siguientes:

|  |
| --- |
| Pasos  CF024\_6.2.\_Bases de datos en tiempo real (2) |

**7. Multimedia**

En Android Studio, la integración de contenido multimedia en las aplicaciones es muy sencilla gracias a las opciones que presenta la API. Las principales clases que facilitan el acceso a los servicios multimedia son *MediaPlayer* y *VideoView,* siendo estas las más utilizadas.

**7.1. *MediaPlayer***

*MediaPlayer* permite reproducir sonido y video a través de su API. Es uno de los componentes principales del marco de trabajo de medios. Un objeto de esta clase puede recuperar, decodificar y reproducir audio y video con una configuración mínima. Es compatible con varias fuentes de medios diferentes, entre las que se incluyen las siguientes: URI, recursos locales y URL externas.

A continuación, se presenta un ejemplo de cómo reproducir un archivo de audio que está disponible como recurso local (ubicado en el directorio **res/raw/** de la *app)*:

|  |
| --- |
| Slide  CF024\_7.1\_MediaPlayer |

**7.2. *VideoView***

La clase *VideoView* permite presentar un archivo de video. Puede cargar videos de varias fuentes (como recursos o proveedores de contenido). La propiedad principal de *VideoView* es **setVideoURI**, donde se establece la ruta del video.

A continuación, se describe un ejemplo para utilizar *VideoView:*

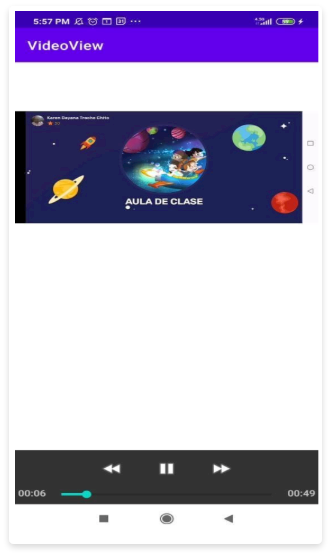
1. Al igual que con el MediaPlayer, se debe crear la carpeta **res/raw** y guardar en esta carpeta el video con extensión .mp4.
2. En el *layout,* se crea el componente *VideoView.*

|  |
| --- |
| <VideoView  android:id="@+id/videoview"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content" /> |

1. En la clase Java se realiza todo el código:

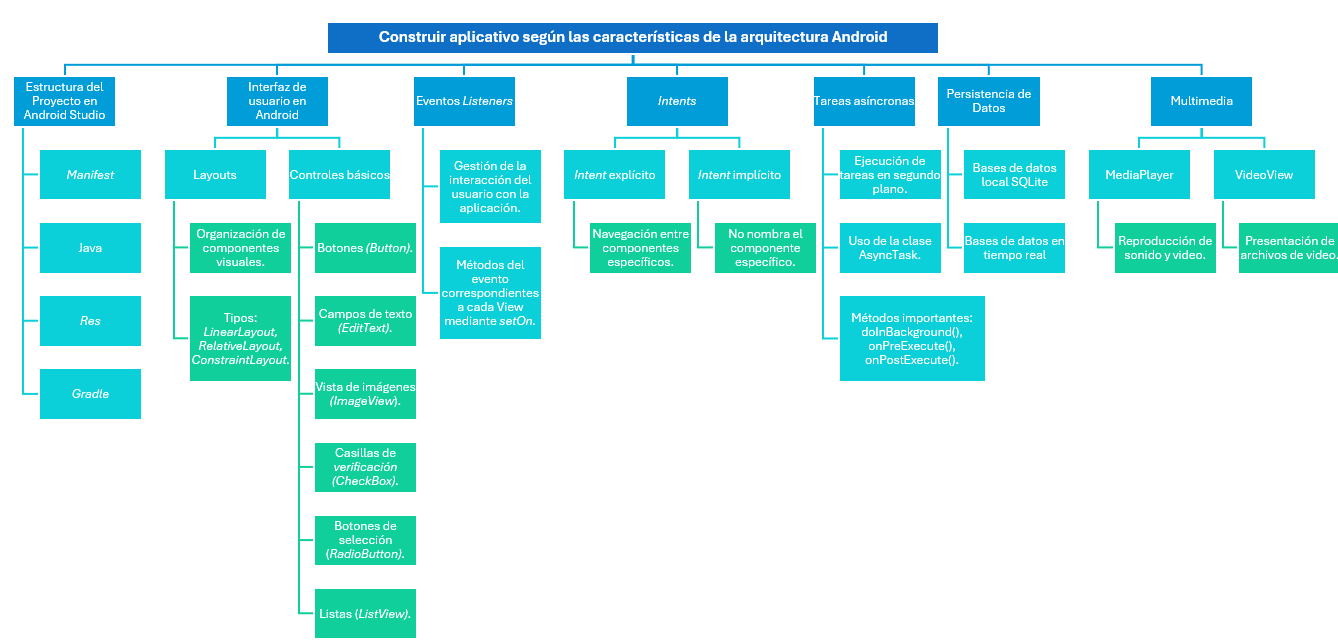
|  |
| --- |
| public class MainActivity extends AppCompatActivity {  // Paso 1. Se declara el VideoView  VideoView videoView;  @Override  protected void onCreate(Bundle savedInstanceState) { // Método principal  super.onCreate(savedInstanceState);  setContentView(R.layout.activity\_main);  // Paso 2. Referenciar el VideoView  videoView = findViewById(R.id.videoview);  // Paso 3. Se asigna la ruta al VideoView  videoView.setVideoPath("android.resource://" + getPackageName() + "/" + R.raw.video);  // 4. La propiedad SetMediaController permite visualizar los controles de pausar, adelantar, iniciar  videoView.setMediaController(new MediaController(this));  // 5. Se inicia la ejecución de video  videoView.start();  }  } |

1. Al ejecutar se presentará de la siguiente manera:
2. Vista Ejecución VideoView



1. **SÍNTESIS**

A continuación, se presenta una síntesis de la temática estudiada en el componente formativo.



1. **ACTIVIDADES DIDÁCTICAS (Se debe incorporar mínimo 1, máximo 2)**

|  |  |
| --- | --- |
| DESCRIPCIÓN DE ACTIVIDAD DIDÁCTICA | |
| Nombre de la Actividad | Desarrollo de aplicaciones nativas para Android |
| Objetivo de la actividad | Comprender los conceptos fundamentales en la creación de aplicaciones nativas para la plataforma Android. |
| Tipo de actividad sugerida | Relacionar conceptos |
| Archivo de la actividad  (Anexo donde se describe la actividad propuesta) | CF024\_Actividad didactica |

1. **MATERIAL COMPLEMENTARIO:**

|  |  |  |  |
| --- | --- | --- | --- |
| Tema | Referencia APA del Material | Tipo de material  (Video, capítulo de libro, artículo, otro) | Enlace del Recurso o  Archivo del documento o material |
| Estructura de un proyecto en Android Studio | Android Developers. (2021, May 25). Aspectos fundamentales de la aplicación. | Página | <https://developer.android.com/training/basics/activity-lifecycle> |
| Estructura de un proyecto en Android Studio | Developers. (2021, June 14). Notas de la versión del complemento de Android para Gradle. | Página | <https://developer.android.com/studio/releases/gradle-plugin> |
| Interfaz de usuario en Android | Developers. (2021) Cómo crear una IU responsiva con ConstraintLayout. | Página | <https://developer.android.com/training/constraint-layout> |
| Interfaz de usuario en Android | Castillo, J. D. L. (2015, May 21). Capítulo 14 - Interacción con Botones, TextView, ImageView. YouTube. | Video | <https://www.youtube.com/watch?v=-z8zTGTo7vA&ab_channel=Jos%C3%A9DimasLuj%C3%A1nCastillo> |
| Interfaz de usuario en Android | Castillo, J. D. L. (2015, May 21). Capítulo 15 - Android Studio Intent Abrir una Activity. YouTube. | Video | <https://www.youtube.com/watch?v=LEVXzsXhO0A&ab_channel=Jos%C3%A9DimasLuj%C3%A1nCastillo> |

1. **GLOSARIO:**

|  |  |
| --- | --- |
| TÉRMINO | SIGNIFICADO |
| Diseño: | un diseño define la estructura visual de una interfaz de usuario. |
| *Scrolling:* | acción de desplazarse internamente. |
| *SDK:* | kit de desarrollo de *software.* |
| *String:* | recurso XML que ofrece una sola *string.* |
| *View:* | es la clase base de los *widgets.* |
| *Wearable:* | dispositivo conectado que se puede llevar puesto y que se conecta al teléfono móvil. |

1. **REFERENCIAS BIBLIOGRÁFICAS:**

Android Developers. (2021). *Documentation | Desarrolladores de Android*. <https://developer.android.com/docs>

1. **CONTROL DEL DOCUMENTO**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Nombre | Cargo | Dependencia  *(Para el SENA indicar Regional y Centro de Formación)* | Fecha |
| Autor (es) | Zulema León | Experta temática | Regional Cauca - Centro de Teleinformática y Producción Industrial | Noviembre 2022 |
| Paola Alexandra Moya | Evaluadora instruccional | Regional Antioquia - Centro de Servicios de Salud | Mayo 2024 |
|  | Olga Constanza Bermúdez Jaimes | Responsable Línea de Producción Antioquia | Regional Antioquia - Centro de Servicios de Salud | Mayo 2024 |

1. **CONTROL DE CAMBIOS**

**(Diligenciar únicamente si realiza ajustes a la Unidad Temática)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Nombre | Cargo | Dependencia | Fecha | Razón del Cambio |
| Autor (es) |  |  |  |  |  |