



Universidad de Valladolid



**ESCUELA DE INGENIERÍAS
INDUSTRIALES**

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

**Grado en Ingeniería de Diseño Industrial y Desarrollo del
Producto**

**Programación de aplicaciones Android para
aprendizaje de Idiomas**

Autor:

Colina Fernández, Andrea

Tutor:

**Escudero Mancebo, David
Departamento de Informática
Área de Ciencia de la Computación
e Inteligencia Artificial**

Valladolid, Junio de 2018.

El punto de partida es una aplicación existente destinada al aprendizaje de español sobre la cual desarrollamos un proceso de rediseño y mejora. Se realiza la aplicación de aprendizaje de idiomas en dos lenguajes de programación distintos: ActionScript 3.0 y Java. Para realizar el desarrollo de versión de la aplicación en Android Studio se lleva a cabo un estudio previo de Programación Orientada a Objetos (OOP). Por otro lado, se utiliza Adobe Animate CC para el desarrollo de la aplicación que utiliza lenguaje AS3. Ambas hacen uso del servicio de Google Speech to Text (STT) para el reconocimiento de voz. Se realiza una comparativa entre ambas aplicaciones, tratando similitudes y diferencias, tanto a nivel gráfico como a nivel de algoritmos y estructura de datos. La aplicación se encuentra disponible para su uso tanto en la tienda Google Play como en App Store.

Based on an existing application destined to learn Spanish language, we make a project of redesigning and improvement it. This project consists of the developing of a learning application of languages in two different programming languages: ActionScript 3.0 and Java. A previous research about Object Oriented Programming (OOP) is made before carrying out the developing of the app in Android Studio. On the other hand, Adobe Animate CC is used for the developing of the application which use AS3 language. Both applications make use of the Google service Speech to Text (STT) for the recognition of voice. Then, a comparative study is made between both applications, the similarities and differences, both at a graphic level as an algorithm and data structure level.

*Aplicación – Interfaz - Programación – Comparativa – Android
Application – Interface – Programming – Comparative Study – Android*

Contenidos

1. Introducción.....	9
1.1. Objetivos:.....	9
1.2. Visión panorámica:	10
2. Programación en JAVA	11
2.1. Programación orientada a objetos	11
2.2. Entorno de Desarrollo.....	11
2.3. Variables.....	12
2.4. Métodos.....	13
2.5. Clases	15
2.6. Estructuras Condicionales.....	17
2.7. Introducir datos al programa	18
2.8. Array y ArrayList.....	20
3. Android.....	23
3.1. Sistema Operativo Android.....	23
3.2. Arquitectura.....	23
3.3. Logotipo e Imagen Corporativa	24
3.4. Android Studio.....	24
3.5. Primeros pasos	25
3.6. Actividades	28
3.6.1. Ciclo de vida de una actividad.....	28
3.6.2. Abrir una actividad: <i>los Intents</i>	29
3.7. Layouts	30
3.8. Widgets	31
3.8.1. TextView y EditText	31
3.8.2. Button.....	32
3.8.3. Radio Button y RadioGroup	33
3.8.4. Los Toasts.....	34
3.8.5. ProgressDialog y ProgressBar	34
3.8.6. Los Threads	35
3.8.7. Reproductor de sonido	35
3.9. Material Design	37
3.9.1. AppBars.....	38
3.9.2. Bottom Sheets.....	39
3.9.3. Coordinated Behaviors	40
3.9.4. Bottom Navigation.....	41

3.9.5.	SnackBars.....	41
3.9.6.	Text Fields.....	42
3.9.7.	Collapsing Toolbars.....	43
3.9.8.	Navigation Views.....	45
3.9.9.	Floating Action Button.....	45
3.9.10.	Tab Layout.....	46
4.	Aplicación.....	49
4.1.	Análisis de las principales aplicaciones para aprendizaje de idiomas.....	49
4.1.1.	Duolingo.....	49
4.1.2.	TinyCards.....	50
4.1.3.	Busuu.....	51
4.1.4.	Memrise.....	52
4.1.5.	Babbel.....	53
4.2.	Diseño de interfaz.....	54
4.2.1.	Primeras ideas.....	54
4.2.2.	Elementos de la interfaz.....	55
4.2.3.	Pantalla de Inicio.....	55
4.2.4.	Pantalla Principal.....	56
4.2.5.	Menú de desbordamiento.....	58
4.3.	Componentes.....	59
4.3.1.	Retos de la pantalla de inicio.....	60
4.3.2.	Retos de la pantalla principal.....	61
4.3.3.	Retos del menú de desbordamiento.....	62
4.3.4.	Acceso a documentos de texto.....	63
4.3.5.	Acceso a fotografías.....	64
4.3.6.	Acceso al audio.....	64
4.3.7.	Uso de ASR – Automatic Speech Recognition Google Speech API.....	65
5.	Implementación (comparativa ActionScript 3.0 vs Android Studio).....	69
5.1.	Proceso de desarrollo en ActionScript 3.0.....	69
5.1.1.	Pantalla de inicio.....	70
5.1.2.	Pantalla Principal.....	73
5.1.3.	Menú de desbordamiento.....	77
5.2.	Reutilización de elementos en Android Studio.....	78
5.3.	Similitudes.....	80
5.3.1.	Similitudes en la interfaz.....	80
5.3.2.	Similitudes en el código.....	80
5.4.	Diferencias.....	83

5.4.1.	Diferencias en la interfaz.....	83
5.4.2.	Diferencias en el código	89
5.4.3.	Diferencias en la navegación entre pantallas.....	93
6.	Usabilidad	95
7.	Conclusiones	97
8.	Bibliografía.....	99

En primer lugar, quiero agradecer a la Universidad de Valladolid la oportunidad de realizar este Trabajo de Fin de Grado, ya que ha supuesto para mí un antes y un después en mi vida como estudiante. A través del estudio y desarrollo de este proyecto he sido consciente de hacia dónde quiero orientar mi futuro laboral, hacia el mundo de la programación.

Para comenzar quiero dar las gracias a mis padres, a mi hermano y a mi familia, que han hecho posible que me encuentre en este momento escribiendo estos agradecimientos. Ellos me han apoyado en todo momento y han sabido tener las palabras adecuadas para impulsarme en cada situación. También quiero darles las gracias a mis amigos, que antes de que yo fuese consciente de qué era lo que realmente me apasionaba ellos lo supieron ver y aconsejarme. Gracias por acompañarme a lo largo de este camino que hemos recorrido juntos y hacerlo más ameno, inolvidable.

Quiero agradecerle en especial a mi profesor y tutor de este proyecto, David Escudero, haber confiado en mí y en mis capacidades desde la asignatura de Técnicas de Presentación Multimedia y después haberlas explotado en este proyecto, sacando lo mejor de mí. Agradecerle también la oportunidad de trabajar en el laboratorio de la investigación de Entornos de Computación Avanzada y Sistemas de Interacción Multimodal ECA-SIMM ya que, sin duda, ha sido una de las mejores experiencias de este curso.

Otro apoyo crucial ha sido contar con el conocimiento y la experiencia de Mario Corrales, en cuanto a Java y Animate CC, y de Cristian Tejedor, desarrollador de aplicaciones Android. Han estado ahí para resolverme cualquier duda y aconsejarme en materias en las cuales son verdaderos expertos. Gracias también por los buenos ratos que hemos pasado en el laboratorio y por aprender observando vuestros proyectos.

También quiero acordarme de mi profesora de informática Margarita Gonzalo, por valorar mi potencial en la asignatura desde la primera semana de carrera. Significó mucho para mí en su momento y lo sigue haciendo hoy en día.

Gracias a este proyecto, pues salgo de la Universidad con dos ofertas de trabajo de grandes empresas del Parque Tecnológico de Boecillo, y un puesto de trabajo, que me ofrece orientar mi carrera profesional hacia lo que me apasiona.

Gracias a cada persona que se ha cruzado en mi camino a lo largo de estos años y que han hecho que hoy sea la persona que soy.

1. Introducción

Cada vez desde más temprana edad, utilizamos el teléfono móvil para aspectos cotidianos de la vida, hasta llegar al punto en que, según El Mundo [15], se “corona al móvil como el principal dispositivo a través del cual los españoles entran en Internet, con un 88,3% de usuarios”. Las empresas son conscientes de que la mejor estrategia digital es aquella que concibe el móvil como una pieza fundamental para llegar al consumidor. Por ello, invierten en el desarrollo de aplicaciones móviles de sus sitios web y en la adaptación web a las plataformas móviles.

La Sociedad Mixta para la Promoción del Turismo de Valladolid ha desarrollado a lo largo de los últimos años programas relacionados con los idiomas y la promoción de la ciudad y provincia. El pasado curso se presenta el Trabajo de Fin de Grado de mi compañera Violeta Portero, el desarrollo de una aplicación destinada al aprendizaje de español tematizada en torno a la ciudad de Valladolid. Este proyecto se presenta a modo de prueba de concepto, por parte del grupo de investigación ECA-SIMM, para su explotación, realizando previamente una serie de modificaciones en cuanto a funcionalidad e interfaz. Este proyecto engloba el trabajo de reingeniería del producto presentado para adecuarse y satisfacer las necesidades que el cliente demanda.

A la hora de pensar en el desarrollo una aplicación, un aspecto fundamental es hacia qué sistema operativo va dirigida. Entre los diferentes sistemas operativos existentes en el mercado, destaca por encima del resto Android, el conocido sistema operativo de Google. Android se define como un sistema operativo de código abierto, esto es, que no solo los fabricantes pueden realizar modificaciones y permite programar de manera gratuita. Según el Diario ABC [22] respecto a las cuotas de mercado: “Android goza del 90%, iOS se queda con el 9.2%, según el último informe de julio de Kantar Worldpanel”. En el mundo, 7 de cada 10 móviles utilizan Android. Es por ello, que resulta interesante orientar el proyecto hacia este sistema operativo.

El programa que se utilizó para el desarrollo de la aplicación original es Adobe Animate CC y en un principio es el que utilizamos para desarrollar la nueva versión. Por otro lado, se estudia la necesidad de cambiar la aplicación al entorno de desarrollo oficial Android Studio. Desarrollar aplicaciones en un entorno Android nos permite hacer uso de las aplicaciones nativas del sistema operativo, como el sistema reconocedor de voz.

1.1. Objetivos:

- Desarrollo del proceso de reingeniería del software, así como del diseño de la aplicación para adecuarla a la promoción de la ciudad de Valladolid como centro de aprendizaje de español.
- Implantación de la aplicación en ActionScript 3.0 y su posterior migración a los principales sistemas operativos para dispositivos móviles: Android, iOS y Windows.
- Migración del código y diseño a Android Studio, previa formación en programación orientada a objetos (OOP), Java y Android Studio.

1.2. Visión panorámica:

- Capítulo 2: Programación en Java

Desarrollo de tutorial de aprendizaje de las nociones básicas respecto a la programación orientada a objetos y Java.

- Capítulo 3: Android

Estudio de la programación de aplicaciones en Android Studio. Estudio de las directrices de diseño recogidas en la guía Material Design.

- Capítulo 4: Aplicación

Estudio de mercado de las principales aplicaciones existentes para el aprendizaje de idiomas. Planteamiento de objetivos, tanto a nivel de código como de diseño, para el desarrollo de la aplicación.

- Capítulo 5: Implementación

Implementación de la aplicación en ActionScript 3.0 y posterior comparativa de similitudes y diferencias con la versión de Android Studio.

2. Programación en JAVA

Como bien he mencionado anteriormente, debemos conocer y entender Java para poder adentrarnos en el mundo de Android SDK. ¿Por qué? Java para aplicaciones Android tiene bastantes puntos en común con el Java que se utiliza para el resto de las aplicaciones. Por tanto, me parece muy importante tener un concepto general de Java para poder luego especializarnos en Java para Android.

2.1. Programación orientada a objetos

Java es un Lenguaje de Programación Orientada a Objetos (POO ó OOP en inglés). La POO innova la manera de tratar los datos, cambiando la concepción que tenemos de C, aunque hay que reconocer que su sintaxis deriva en gran medida de C y C++. Este tipo de programación trata los datos de entrada a partir de la creación de objetos, y a través de ellos obtiene unos datos de salida específicos. Para entender mejor Java, a continuación, vamos a definir una serie de conceptos que utilizaremos a lo largo de todo el capítulo.

Conceptos Fundamentales:

- Clase <Class>: Veamos este concepto con un ejemplo: clase Coche. Dentro de esa clase se definen todas las características genéricas, así como los comportamientos de un objeto en concreto. Por ejemplo, para la clase Coche: número de ruedas, marca, modelo, color ...
- Objeto <Object>: Es la instancia de una clase. Siguiendo con el ejemplo anterior, un objeto de tipo coche tiene una serie de características como de qué marca es, cuál es el modelo, el color... Todas esas características, o atributos, son los que distinguen ese objeto de otro cualquiera. Un objeto tiene unos atributos (datos) y unos comportamientos (métodos) particulares de una clase.
- Método <Method>: Es un algoritmo que se usa asociado a un objeto. Su reacción se desencadena tras la recepción de un mensaje. Coloquialmente se podría decir que es aquello que puede realizar un objeto para producir un cambio de estado en sus propiedades o generar un evento.
- Evento: Se define como un suceso del sistema, reacción desencadenada por un objeto.

2.2. Entorno de Desarrollo

Una pregunta importante sería ... ¿Dónde escribo mis líneas de código para probar todo lo que voy a aprender? Pues bien, existen multitud de plataformas de desarrollo. En concreto, vamos a trabajar con Eclipse, que es entorno integrado de desarrollo (IDE) de código abierto basada en Java. El entorno de trabajo de Eclipse es mucho más acogedor y sencillo de utilizar que la Terminal donde solíamos programar en C. Eclipse nos muestra donde tenemos los errores y nos da posibles soluciones. Realiza las acciones de una manera más visual para el programador.

Descargar Eclipse: <https://www.eclipse.org/downloads/>

2.3. Variables

A continuación, trataremos un poco sobre los tipos de datos más básicos en Java y su declaración en el código.

Tipos de Datos:

Como en C, trabajamos definiendo tipos de datos. Dependiendo de la naturaleza de ese dato, necesitaremos un tipo u otro. Para definir correctamente las variables de nuestro código debemos utilizar las palabras clave.

int – Número entero (sin decimales). Se incluye el cero, números positivos y negativos.

float – Incluye tantos decimales como pueda contener. El número de decimales se entiende como variable, y por tanto hay que tener en cuenta que este número es impreciso. Cuando es necesaria precisión se debe utilizar otro tipo de datos que es BigDecimal.

boolean – Es un bit que devuelve true o false. “True”=1, “False”=0.

char – Representa un único caracter. Distingue símbolos (&,# ...), mayúsculas y minúsculas.

String – Cadena de caracteres. Típicamente es lo que entendemos como palabras, también puede almacenar números, pero a menos que los convirtamos a otro tipo, no podremos operar con ellos. Los espacios en blanco cuentan como caracteres.

*Las variables de tipo int, float, boolean y char son tipos de datos primitivos. Se conciben como tipos de datos sencillos. La variable String lleva “S” mayúscula porque es un tipo de dato complejo. Para Java String son objetos, ya que ,por ejemplo, se puede llamar al método length() que nos dice cuantos caracteres tiene nuestra cadena tipo String.

Definición de Variables:

En este sentido, seguimos teniendo una sintaxis bastante parecida a C. Para definir una variable se sigue siempre el mismo esquema, dependiendo del tipo que sea.

Ejemplos: int numeroRuedas;
 float litrosGasolina;
 boolean sufucienteGasolina=true;
 char letra1Matricula = 'V';
 String propietario = “Juan González Pérez”;

*Las variables inicializadas son aquellas a las que se les asigna un valor.

Para definir una variable de tipo String siempre se sigue la misma sintaxis:

String nombreVariable = “ _____ ” ;
 1 2 3 4 5

1. String – tipo de dato

2. nombreVariable – típicamente para los nombres de las variables en general, la primera letra es minúscula y no se admiten espacios ni caracteres especiales.
3. (=) equals – operador de asignación.
4. “___” – La cadena de caracteres que queremos almacenar en el String se debe colocar entre las comillas (“”)
5. ; semicolom – igual que en C, en Java se utiliza siempre al final de la línea de código.

Tipos de Variables:

Podemos declarar las variables según donde estén ubicadas. Las variables pueden ser de tipo *Static* o *Class Variable*. Este tipo de variables van asociadas a la clase, así como a todas sus instancias. Por ejemplo, si creamos una variable estática dentro de una clase y luego creamos tres objetos referidos a esa clase. Al cambiar en uno de ellos el valor de la variable estática, automáticamente se cambiará en los demás objetos.

En segundo lugar, tenemos *Instance Variable*. A diferencia de la anterior, este tipo de variables tienen su propia copia particular. De este modo, los cambios realizados sobre alguna propiedad del objeto, solo se verán reflejados sobre ese objeto.

Por últimos tenemos las *Variables Locales*. Éstas son declaradas en el interior del método en cuestión, y no se pueden utilizar desde el exterior del mismo. En este sentido, las variables de dentro de los métodos pueden tener el mismo nombre que las declaradas en el main o en las clases, que no supondrá ningún problema.

2.4. Métodos

Los métodos, como dijimos anteriormente, son aquella parte del código que nos devuelve una acción o algún tipo de resultado. Los métodos podrían ser concebidos como partes reutilizables de código, ya que podemos llamar a ese método cuantas veces queramos sin tener que reescribirlo cada vez que lo utilicemos.

Para declarar un método podemos seguir el siguiente esquema:

```
public/private      dataTypeReturn  nombreMétodo ( ) { }
    1                2                3      4  5
```

1. public/private definición del método. Si se define private no se podrá utilizar desde fuera de la clase en donde esté declarado. Un método declarado como public se puede llamar desde donde sea.
2. Data type return – tipo de dato que devuelve el método. Por ejemplo, si tenemos un método que devuelve si el depósito está lleno o no, debemos declarar boolean. Además añadiremos: return variableTipoBoolean; que debe estar declarada previamente. Si el método no devuelve nada, será de tipo void. No lleva return.
3. nombreMetodo: El nombre del método debe ir en minúsculas sin espacios.

4. () paréntesis: Los paréntesis acogen la declaración de las variables de entrada al método. Por ejemplo, si queremos pasar los litros de gasolina a galones, debemos poner: `public int litrosGalones (int litrosGasol){ ... }`, con la variable `litrosGasol` opera el método dentro y en la llamada al método debemos introducir la variable que almacena ese dato: `coche.litrosGalones (litrosGasolina)`; de modo que dentro del método `litrosGasol` coge el valor de `litrosGasolina` de fuera del método.
5. { } corchetes: dentro de los corchetes se ubica el código del método. Ahí es donde se opera con las variables y realiza las acciones necesarias. Dentro de los corchetes debemos colocar el 'return' que mencionamos antes en caso de necesitarlo el método.

Antes he mencionado la llamada al método, pero... ¿Qué es?

Cuando *llamamos al método* le estamos diciendo al programa que utilice esa parte de código que escribimos fuera en un sitio concreto. El método podemos llamarlo con relación a un objeto o no, en el segundo caso debemos declarar *static* justo después de `public/private` .

Ejemplo de método y su llamada: `leerLibros()` Introduce los datos de N libros y los almacena en un vector declarado fuera del método.

Entrada: –

Salida: – (no ponemos `return- void-`)

```
public static void leerLibros() {
    String titulo, autor,texto;
    texto = "";
    int ISBN,numPaginas,numLibros;

    texto =JOptionPane.showInputDialog("Introduce un número de
libros: ");
    numLibros= Integer.parseInt(texto);

    for( int i=0;i<numLibros;i++) {
        titulo = JOptionPane.showInputDialog( "Título del libro: ");
        texto = JOptionPane.showInputDialog( "ISBN");
        ISBN = Integer.parseInt(texto);
        autor = JOptionPane.showInputDialog( "Autor: ");
        texto = JOptionPane.showInputDialog("Número de páginas: ");
        numPaginas = Integer.parseInt(texto);

        Libro aux = new Libro();

        aux.setTitulo(titulo);
        aux.setAutor(autor);
        aux.setISBN(ISBN);
        aux.setNumPaginas(numPaginas);

        Libros.add(aux);
    }
}
```

Ésta es la función `main`, desde donde llamamos a nuestro método.

```
public static void main(String[] args) {
    LeerLibros(); }
}
```

A continuación, veremos un ejemplo de una llamada a un método en relación a un objeto. Este programa crea un objeto tipo Coche (clase que hemos definido anteriormente), que se llama Seat. Con el método ruedas nos devuelve que el coche tiene cuatro ruedas. El dato que devuelve el método tiene que estar referido en la clase 'main' a un dato declarado del mismo tipo.

Entrada: --

Salida: número de ruedas.

```
public int ruedas(){
    ruedas= 4;
    return ruedas;
}

public static void main(String[] args) {
    Coche Seat = new Coche();
    int numRuedas= Seat.ruedas();
}
```

2.5. Clases

Una Clase en Java se concibe como un grupo de instrucciones acerca de cómo es la estructura de un conjunto de datos. Ahora veremos un pequeño esquema de una clase genérica en Java.

```
public/private class Coche { //Nombre de la clase en mayúsculas
    private String matricula; //Definición de los atributos
    private int numRuedas;
    private String marca;
//Constructor de clase*
//Operaciones disponibles **
}
```

Constructor de clase – Un constructor es aquella parte del código (método, función ...) que nos dice cómo van a ser los objetos que creamos de nuestra clase, es decir, el estado inicial del objeto. Si no existe constructor, el compilador añade uno público que no tiene ninguna función. Al ser métodos, los constructores también aceptan parámetros. Podemos señalar unas características generales a todos los constructores:

- El constructor tiene el mismo nombre que la clase a la que pertenece.
- No pueden ser heredados.
- No retorna ningún valor, ni siquiera es 'void'. En la declaración no podemos especificar ningún tipo de dato.
- Normalmente, se declara como 'public'.

Operaciones disponibles – Las operaciones disponibles en una clase se almacenan como métodos dentro de la misma. Aquí es donde se almacenan los métodos setters y getters. ¿Para qué sirven? Pues bien, ya hemos creado nuestra clase Coche, y queremos acceder a algún dato particular del objeto que hemos creado a través de un constructor. Sin los métodos que hemos mencionado antes sería imposible, ya que las variables están declaradas como 'private' y no es posible acceder a la información.

Método setter: set (del inglés) significa asignar, establecer. Este método nos permite asignar un valor a un atributo, por tanto, es 'void', ya que no retorna nada. Con este método damos acceso público a los diferentes atributos del objeto que el usuario desee modificar.

Una declaración común de un método setter podría ser la siguiente:

```
public void setMarca(String marca){  
    this.marca=marca;  
}
```

Método getter: get (del inglés) significa obtener. Este método nos permite obtener el valor asignado anteriormente, que de la misma manera que en los setter, es información que está en private y de otra manera no podríamos acceder. El método getter, a diferencia del setter, es de la naturaleza de la variable que debe devolver. Ambos métodos se declaran como public.

Una declaración común de un método getter sería:

```
public String getMarca() {  
    return marca;  
}
```

Todas las variables declaradas en una clase deben de tener sus métodos setter y getter para acceder a ellas y modificarlas.

Modificadores de Acceso – Haciendo un pequeño inciso, me gustaría tratar un poco acerca de los *Modificadores de Acceso*. A lo largo de este capítulo llevamos hablando de que un método es public, otra variable es private ... Realmente ¿sirve para algo?

Tanto para las clases, como para los métodos y variables, debemos definir el acceso, es decir, quién puede acceder a ellos. En ciertos casos necesitaremos que las cosas sean públicas y estén disponibles para el que las necesite. Por el contrario, si necesitamos proteger los datos con respecto al exterior, debemos ponerlo en private.

En relación con lo anterior, la base de la encapsulación es ésta: mostrar mediante los métodos setter y getter los atributos que realmente sean necesarios, mientras que el resto se protegen con private o protected.

Las palabras clave para realizar todo esto son: public, private y protected.

public: Da acceso publico desde cualquier lugar.

private: Acceso solo dentro de la clase.

[vacío]: Acceso de paquete.

protected: Acceso desde la clase y sus hijos en "herencia".

2.6. Estructuras Condicionales

Recordando lo aprendido en C, las estructuras condicionales que vamos a utilizar son básicamente las mismas.

a. Condición 'if':

Sintaxis: `if(condición){ }`

Si se cumple la condición que está en el interior de los paréntesis se realiza lo que hay dentro del bucle.

b. Condición: if – else { }

Sintaxis: `if(condición){
}else { }`

Si no se cumple la condición del if, el programa pasa a leer lo que hay dentro de los paréntesis de else. En esta estructura suele ser bastante común verla anidada, es decir una dentro de otra. Ejemplo:

```
if (cont>0) {  
    cont--;  
}else {  
    cont=0;  
}
```

c. Bucle 'for':

Sintaxis: `for (condición inicial; condición final; condición de paso) { }`

La condición inicial nos indica qué tiene que suceder para que se inicie el bucle.

La condición de paso es aquella que permite que el bucle fluya (`i + +` ó `i - -`)

La condición final es la que hace que el bucle no sea infinito y, por tanto, nos de error.

Ejemplo: (Recorre todos los datos almacenados en un vector y los imprime en pantalla)

```
for(int i =0;i<vec.length;i++) {  
    System.out.println(vec[i]+" \n");  
}
```

d. Bucle 'while':

Sintaxis: `while(condición){ condición de paso }`

Dentro de los corchetes, debe existir una condición de paso, es decir, una condición que actualice el bucle. Mientras que se cumpla la condición de while, el bucle seguirá funcionando. En el momento que no se cumpla, el compilador sale del bucle a la siguiente línea.

e. Bucle 'do – while':

Sintaxis: `do{
}while(condición);`

Con este bucle nos aseguramos que, aunque la condición de while no se cumpla, se realizará al menos una vez lo que esté dentro de los corchetes.

f. Bucle 'for-each':

Sintaxis: `for(typeData nameVar : Array){ }`

La variable 'nameVar' es una variable auxiliar que almacena cada elemento del vector. Lo recorre entero. Debemos especificar el tipo de datos que hay en el vector. Se puede leer como "Para cada elemento del vector" realizará la acción situada entre los corchetes. Ejemplo:

```
for(String marca : arrayCoches) {  
    system.out.println(marca);  
}
```

g. 'switch':

Sintaxis:

```
switch(variable){  
    case valor1:  
        //código  
        break;  
    case valor2:  
        //código  
        break;  
    case valor3:  
        //código  
        break;  
    default:  
        //código  
        break;  
}
```

El switch permite realizar diferentes operaciones en función del valor de una variable. Si se da la situación de que no se cumpla el valor de ninguno de los 'case' se pasa a 'default'.

break; es un indicador de salto. En caso de que se lea, por ejemplo, la opción valor2 y no haya break, seguiría luego con el resto. Break; hace que lea esa opción y se salga del switch.

h. Condicional Yes/No

Sintaxis:

```
(condición? respYes :respNo);
```

Se declara una condición a cumplir. Si se cumple se realiza la opción situada en respYes (respuesta sí), sino se realiza respNo.

2.7. Introducir datos al programa

En ocasiones, por ejemplo, cuando creamos objetos, introduciremos los datos mediante los constructores. En otros casos, necesitaremos introducirlos a través del teclado al programa.

En Java hay diferentes maneras de comunicarse con el programa. A continuación, veremos dos de ellas que son bastante frecuentes.

A. Scanner – La clase Scanner nos facilita bastante la manera de obtener datos. Esta clase no sólo nos permite obtener datos a través del teclado, sino también desde archivos.

El usuario quiere que el programa recoja un dato, que es el que está pidiendo entre los corchetes. El programa recogerá ese dato mediante el método next(); almacenándolo en una variable que hemos declarado previamente. Debemos especificar en el método qué tipo de dato se espera, ya que si no coincide el tipo de dato con el dato de entrada dará error (por ejemplo, se espera String y se introduce int).

Ejemplos:

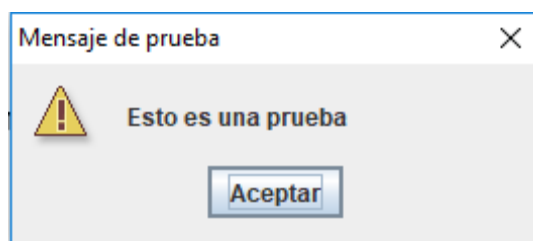
<code>String myString = next ();</code>	Lee hasta donde encuentra un espacio.
<code>String myString2= nextLine();</code>	Lee hasta el final de la línea.
<code>int num1= nextInt();</code>	Lee un número entero.
<code>float num2= nextFloat();</code>	Lee un número de tipo flotante.
<code>double num3= nextDouble();</code>	Lee un número de tipo double.

B. `JOptionPane` – Otra forma muy común de interactuar con el programa es mediante los cuadros de diálogo. El objeto que nos permite realizar esta acción no es otro que `JOptionPane` y, dependiendo de los métodos que se utilicen realizaremos diferentes opciones.

Para comenzar a utilizarlo primeramente debemos importar la clase (como hicimos con `Scanner`): `import javax.swing.JOptionPane;`

- Método: `showMessageDialog(parametro1, parametro2, parametro3,parametro4);`
 - o Parámetro 1: componente padre sobre el cuál se mostrará el mensaje. Normalmente suele ser "null".
 - o Parámetro 2: Mensaje que queremos mostrar en el cuadro de diálogo.
 - o Parámetro 3: Mensaje que se colocará en el título de la barra (puede omitirse).
 - o Parámetro 4: Esta variable almacena el icono que se mostrará en la ventana, por ejemplo, `JOptionPane.WARNING_MESSAGE` (puede omitirse y saldrá una imagen por defecto).

Código: `JOptionPane.showMessageDialog(null,"Esto es una prueba","Mensaje de prueba",
JOptionPane.WARNING_MESSAGE);`

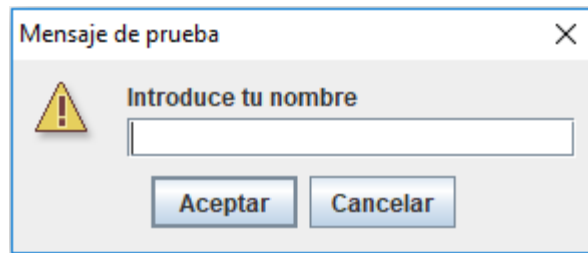


1. Cuadro de diálogo

- Método: `showInputDialog(parametro1, parametro2, parametro3,parametro4);`
- Parámetro 1: Componente padre (null)
- Parámetro 2: Texto del cuadro de mensaje.
- Parámetro 3: Mensaje del título del cuadro.
- Parámetro 4: Imagen que se mostrará en el cuadro.
- (Existen diferentes variantes de los parámetros de este cuadro)

Código: `JOptionPane.showInputDialog(null,"Introduce tu nombre","Mensaje de prueba",`

```
JOptionPane.WARNING_MESSAGE);
```



2. Cuadro de introducción de datos

Aquí se espera recoger un dato del teclado. Por tanto, para utilizar correctamente este método debemos declarar una variable de tipo String para que recoja el dato. (Nota: casi todos los métodos devuelven String)

```
String myName= JOptionPane.showInputDialog(null,"Esto es una prueba...", ...);
```

En caso de que queramos introducir otro tipo de dato, debemos convertirlo al tipo que necesitamos (por defecto el dato es String) Para ello tenemos el método `parse(variable);`. Tenemos que especificar el tipo de dato al que queremos convertir el texto.

Veámoslo mejor con un ejemplo:

```
int myAge;  
String aux = JOptionPane.showInputDialog(null,"Introduce tu edad", ...)  
myAge = Integer.parseInt(aux);
```

2.8. Array y ArrayList

A continuación, vamos a explicar brevemente el concepto de Array y ArrayList. El primero puede que nos suele familiar por su similitud con C.

- A. Array – Un array nos permite almacenar un número fijo de datos que guardan una relación entre sí, es decir, que son de una misma clase. A cada elemento del Array se accede a través de un índice que identifica su posición. La numeración de los índices va desde 0 a n-1, siendo la longitud del Array de 'n' datos.

Para utilizar un Array lo primero que necesitamos es declararlo: `dataType[] nombreArray;`

```
int[] alturasClase = new int[5]; (creamos un array de 5 datos de tipo int)
```

Para inicializar el array basta con situarse en el índice de la posición que deseemos y asignarle un valor:

```
alturasClase[3]= 1.75;
```

```
alturasClase[1]=1.52;
```

```
alturasClase[0]=1.97; (Recordamos que los índices van de 0 a n-1, en este caso, hasta 4)
```

- B. ArrayList – La clase ArrayList, de la misma manera que los Array, permiten almacenar información, pero a diferencia de los otros, lo hace de manera dinámica. Esto es, que no es necesario declarar su tamaño como hacemos en los Array.

Para crear un ArrayList debemos declararlo: `ArrayList<dataType> nameArrayList = new ArrayList<><dataType>;`

```
ArrayList<String> marcasCoche = new ArrayList<String>();
```

Algunos métodos que podemos utilizar con un ArrayList son:

- `size();` Devuelve el número de elementos del ArrayList.
- `add(dato);` Añade el dato en la primera posición libre del ArrayList.
- `add(i,dato);` Añade el dato en la posición i.
- `get(i);` Devuelve el valor del dato de la posición i.
- `remove(i);` Borra el dato que hay en la posición i. Los demás datos se recolocan.
- `set(i, dato);` Coloca el dato en la posición i. Si hay un dato lo sustituye.

Los métodos se declaran llamando al ArrayList seguido del método:

```
marcasCoche.add("Seat");
```

```
marcasCoche.add("BMW");
```

```
System.out.println(marcasCoche); [Seat,BMW]
```


3. Android

3.1. Sistema Operativo Android

Para poder comenzar a trabajar en Android, es necesario situarse en contexto y conocer un poco su historia para poder comprenderlo mejor. Hoy en día, Android es un concepto bastante conocido para muchos de nosotros. Sabemos que es un sistema operativo que se basa en núcleo Linux y su diseño está orientado principalmente a dispositivos electrónicos de pantalla táctil. Android Inc. es la empresa encargada del desarrollo del software ayudada del respaldo económico de Google, empresa que adquirirá Android Inc. en 2005. Los dispositivos Android están muy generalizados en el mercado, tanto que tienen una cuota de 92,2%.

3.2. Arquitectura

Las estructuras del sistema operativo Android se componen de aplicaciones ejecutadas en un marco de trabajo Java de aplicaciones orientadas a objetos. Las aplicaciones se ejecutaban en una máquina virtual Dalvik compiladas en tiempo de ejecución, hasta la versión 5.0, que se cambia al entorno Android Runtime.

Android se compone de:

Aplicaciones – Como hemos mencionado anteriormente, todas ellas están programadas en Java. Se incluyen desde las más elementales como Correo Electrónico, Calendario, SMS ... a otras más complejas y específicas.

Marco de trabajo de las aplicaciones – La arquitectura de este sistema operativo se orienta a la capacidad de reutilización del mayor número de componentes. Una aplicación puede hacer uso de las capacidades de otra. Los desarrolladores tienen acceso completo a las mismas API¹ del entorno de trabajo usadas por las aplicaciones base.

Bibliotecas – Incluye una serie de bibliotecas C/C++ usadas por varios componentes del sistema. Las bibliotecas se exponen a los desarrolladores a través del marco de trabajo de Aplicaciones Android.

Runtime de Android – Android incluye una serie de bibliotecas base que proporcionan la mayor parte de las funciones disponibles en las bibliotecas de Java. Cada aplicación Android corre su propio proceso.

Núcleo Linux – Android es un sistema dependiente de Linux en cuanto a los servicios base de los sistemas de seguridad, gestión de la memoria, de los procesos, ... El núcleo Linux actúa como una capa de abstracción².

¹ API – Abreviatura de “Interfaz de Programación de Aplicaciones”. Esta interfaz nos permite un acceso restringido a la base de datos de un servicio web, protegiendo la información.

² Capa de abstracción – Elemento del sistema operativo que permite acceder de una forma clara al sistema de datos. Actúa como una interfaz entre el software y el hardware.

3.3. Logotipo e Imagen Corporativa

Como estudiante de Diseño Industrial, me parece importante analizar la imagen corporativa de Android, ya que se ha consagrado como una de las imágenes de la década permaneciendo de manera atemporal.

Respecto a su origen etimológico, el término Android, así como Nexus One, hacen referencia a la novela de Philip Dick *¿Sueñan los androides con ovejas eléctricas?*, que posteriormente se lleva a la gran pantalla bajo el nombre de Blade Runner.

El logotipo de Android (ese pequeño androide verde) es Andy, el robot del libro. Existen otras muchas conjeturas por su similitud con R2D2, uno de los personajes de la Guerra de las Galaxias o con el personaje del videojuego "Gauntlet: The Third Encounter".



1. Logotipo de Android

La tipografía del logotipo es *Norad*, El resto del sistema Android trabaja con una familia tipográfica creada por Ascender Corporation llamada *Droid*.

Color: el color de Android se define como "amarillo verdoso" y lo bautizan bajo el nombre de Android verde. En los encabezados y otros formatos el color varía notablemente.

- Valor hexadecimal RGB: # A4C639
- Pantone: PMS376C

3.4. Android Studio

En el capítulo anterior vimos de forma muy básica unas nociones de Java que nos permitirán programar en Android Studio de manera más fluida. Las aplicaciones se desarrollan en Java usando el Android Software Development Kit, más conocido como Android SDK. Aunque estén disponibles otros kits de programación, como el Google App Inventor, nosotros hemos decidido utilizar el entorno de trabajo de Android Studio porque es perfecto para programadores novatos en este campo. Para desarrollar aplicaciones para Android no se requieren grandes conocimientos de lenguajes de programación. Los requisitos son: tener un conocimiento aceptable de Java y estar provisto del Kit de Desarrollo (que está disponible de forma gratuita en Google).

Las aplicaciones Android, como veremos posteriormente, están comprimidas en formato *.apk, lo que permite que sean fácilmente instalables en cualquier dispositivo.

A la hora de comenzar a programar nuestra aplicación debemos conocer que cada versión de Android tiene un nombre en clave y un nivel de API. Por lo general, se intenta dar cabida al mayor número de API's posible. El nivel de API especifica el número de librerías disponibles para desarrollar una aplicación. (Por ejemplo: Versión 4.4 | Nombre en clave: KitKat | API's: nivel 19). A la hora de desarrollar nuestra aplicación debemos incluir una versión mínima (minSdkVersion) de API que nos permita que nuestra aplicación funcione. También debemos incluir una versión máxima (maxSdkVersion). Con estos dos parámetros definimos en qué dispositivos funcionará nuestra aplicación.

En Android Studio podemos encontrar los siguientes componentes:

- Entorno integrado de desarrollo (Android Studio)
- Android SDK Tools: se incluyen un conjunto de herramientas de desarrollo y depuración de programas.
- Android Platform Tools: son las herramientas necesarias para poder compilar y generar los *.apk para Android.
- La imagen que presenta el sistema operativo Android.

Debemos dejar claro, antes de embarcarnos en nuestra primera aplicación, que no se programa de la misma forma para un desktop (escritorio) que para un smartphone. En un smartphone el teclado y la pantalla son mucho más pequeños, la capacidad de señalar objetos es menos precisa y más torpe. También tenemos que señalar que la capacidad de procesamiento de un teléfono es inferior, así como la conectividad a Internet, independientemente de la cobertura que posea.

3.5. Primeros pasos

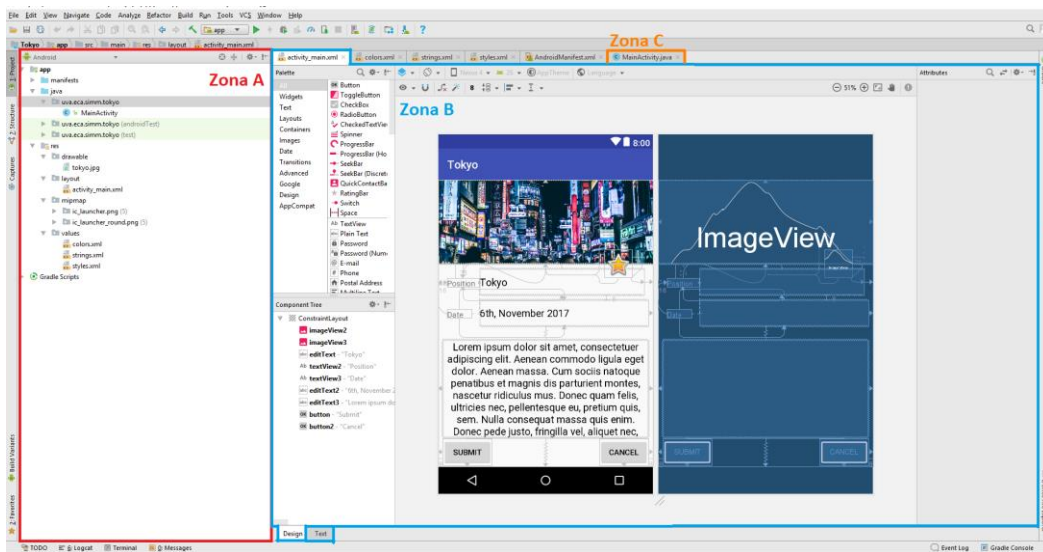
Para comenzar con Android Studio debemos crear un proyecto nuevo: File/New/New Project...

En la primera pantalla nos pide el nombre de nuestra aplicación, que debe empezar por mayúsculas, puede contener espacios, pero no caracteres de tipo ASCII. También nos pide el dominio de la compañía. Por último, especificamos donde queremos guardar nuestro proyecto.

A continuación, debemos escoger para qué tipo de dispositivos va dirigido nuestro proyecto: Phone and Tablets, Wear, TV o Android Auto. Debemos aplicar el nivel de API. Recordamos que cuanto más bajo el nivel de API más compatibilidad tendremos con los dispositivos. El programa nos indica el % de compatibilidad.

En la siguiente pantalla nos indica que seleccionemos un tipo de activity entre los diferentes modelos disponibles. Una "Activity" podríamos decir que es nuestro programa en sí mismo y contiene la interfaz de usuario de nuestra aplicación. Es un programa pequeño y ligero sometido y controlado por Android. Seleccionaremos 'Empty Activity' (vacía) para comenzar.

A continuación, se abre el entorno de trabajo de Android Studio, lo vamos a dividir en tres zonas:



2. Entorno de trabajo de Android Studio y zonas

- Zona A: El explorador de proyectos a la izquierda. En el desplegable Android, nos encontramos las tres carpetas principales: manifests, java y res.
 - Carpeta *manifests*: contiene la descripción del fichero que estamos creando, así como los componentes que están incluidos y los permisos. Está escrito en XML.
 - Carpeta *java*: en esta carpeta encontraremos todo el código fuente de la aplicación escrito en Java. Por defecto, el programa crea un archivo, que en nuestro caso se llama MainActivity.java, donde estará el código de nuestra ActivityMain.
 - Carpeta *res*: Aquí es donde se almacenan todos los ficheros y recursos necesarios para el correcto funcionamiento del proyecto. Dentro nos encontramos con cuatro carpetas principales:
 - *Drawable*: Contiene las imágenes y elementos gráficos usados en la app.
 - *Mipmap*: Contiene los iconos de la aplicación en sus diferentes resoluciones.
 - *Layout*: Aquí se ubican los ficheros de definición en XML de las pantallas de la interfaz gráfica. Se pueden definir diferentes layouts en función de la orientación de la pantalla.
 - *Values*: Encontramos varios ficheros, el fichero XML de definición de los colores de la aplicación(colors.xml), el fichero XML de las cadenas de texto utilizadas(strings.xml) y por último el fichero utilizado para referenciar los estilos utilizados en la aplicación(styles.xml).

En la zona central superior, podemos ver dos pestañas. Una tiene extensión .java y la otra .xml

- Zona B: si nos situamos en la pestaña .xml veremos una zona de diseño de la interfaz de usuario.

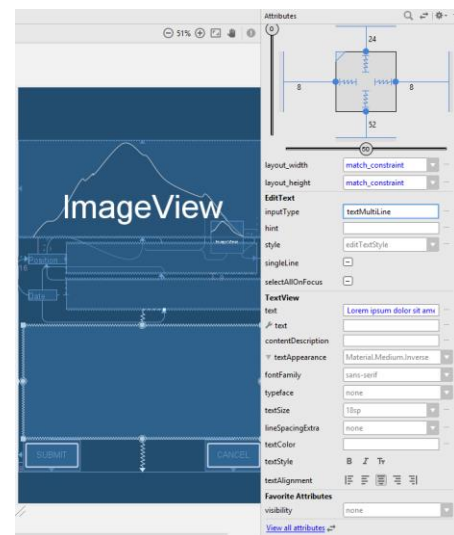
Abajo, en la ventana de diseño tenemos dos pestañas: “Design” y “Text”.

- **Design:** En la parte izquierda tenemos una serie de layouts, widgets y elementos. Posteriormente explicaremos los más importantes. En la zona central tenemos el diseño de la pantalla, donde podemos arrastrar los objetos y colocarlos a nuestro gusto. Tenemos que darles una serie de cotas de posición para evitar que se descoquen al compilar la aplicación. También podemos escoger la orientación de la pantalla, o diseñar diferentes entornos en función de la orientación de la misma.
- **Text:** En esta zona es donde va el código XML. Se autogenera por Android. Podemos cambiar las propiedades de los elementos que componen nuestra aplicación, que también podemos hacerlo desde el entorno de Design. Es más sencillo utilizar la interfaz de Android para definir las características de cada componente, pero yo personalmente creo que es más útil y recomendable utilizar las definiciones XML.

```

activity_main.xml | colors.xml | strings.xml | styles.xml | AndroidManifest.xml | MainActivity.java
1 <?xml version="1.0" encoding="utf-8"?>
2 <android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:app="http://schemas.android.com/apk/res-auto"
4     xmlns:tools="http://schemas.android.com/tools"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     tools:context="uva.eca.simm.tokyo.MainActivity">
8
9     <ImageView
10        android:id="@+id/imageView2"
11        android:layout_width="0dp"
12        android:layout_height="150dp"
13        android:adjustViewBounds="true"
14        android:cropToPadding="false"
15        android:scaleType="centerCrop"
16        app:layout_constraintLeft_toLeftOf="parent"
17        app:layout_constraintRight_toRightOf="parent"
18        app:layout_constraintTop_toTopOf="parent"
19        app:srcCompat="@drawable/tokyo"
20        tools:layout_constraintRight_creator="1"
21        tools:layout_constraintLeft_creator="1" />
22
23     <ImageView
24        android:id="@+id/imageView3"
25        android:layout_width="50dp"
26        android:layout_height="50dp"
27        app:srcCompat="@android:drawable/btn_star_big_on"
28        tools:layout_constraintTop_creator="1"
29        tools:layout_constraintRight_creator="1"
30        tools:layout_constraintBottom_creator="1"
31        app:layout_constraintBottom_toBottomOf="@+id/imageView2"
32        android:layout_marginEnd="21dp"
33        app:layout_constraintRight_toRightOf="@+id/editText"
34        app:layout_constraintTop_toBottomOf="@+id/imageView2"
35        android:layout_marginRight="21dp" />
36
37     <EditText
38        android:id="@+id/editText"
39        android:layout_width="0dp"
40        android:layout_height="47dp"
41        android:layout_marginEnd="16dp"
42        android:layout_marginRight="16dp" />

```



4. Pantalla Text

3. Pantalla Design

- **Zona C:** nos situamos en la pestaña *.java en la parte superior central. Aquí es donde programaremos los comportamientos de los componentes definidos en la Zona B. Esta es la zona

```

activity_main.xml | colors.xml | strings.xml | styles.xml | AndroidManifest.xml | MainActivity.java
1 package uva.eca.simm.tokyo;
2
3 import android.support.v7.app.AppCompatActivity;
4 import android.os.Bundle;
5
6
7 public class MainActivity extends AppCompatActivity {
8
9     @Override
10    protected void onCreate(Bundle savedInstanceState) {
11        super.onCreate(savedInstanceState);
12        setContentView(R.layout.activity_main);
13    }
14 }

```

5. Pantalla Java

de código Java. Tendremos tantas pestañas de java como clases o activities creemos. Más adelante aprenderemos a relacionar unas pestañas con otras.

Para entenderlo de una manera global, en XML se declaran los componentes de la aplicación mientras que en Java se definen los comportamientos.

3.6. Actividades

3.6.1. Ciclo de vida de una actividad

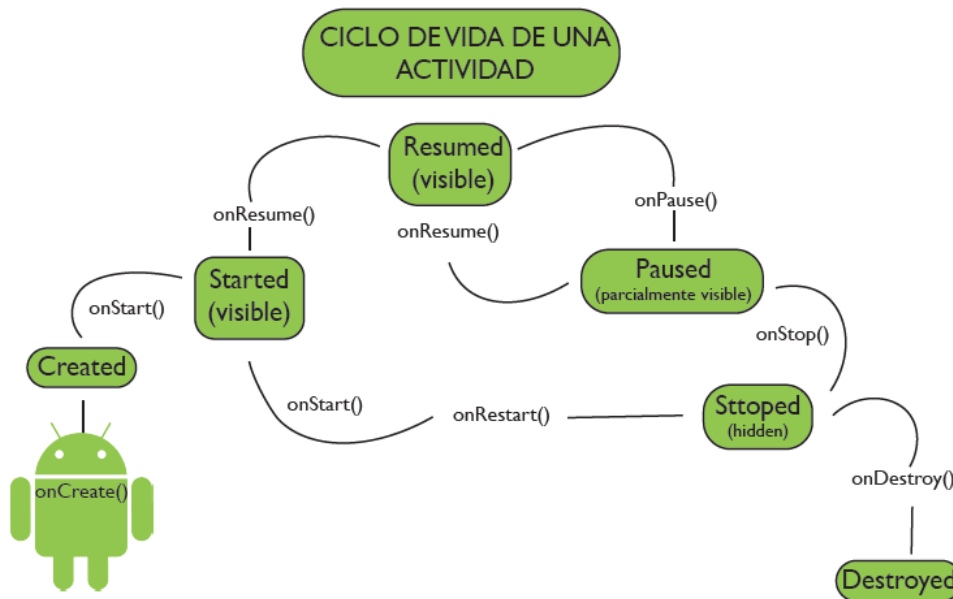
Las activities de nuestra aplicación pueden encontrarse en diferentes estados, dependiendo del momento y la función que desempeñen. Las activities pueden encontrarse en diferentes estados:

- Activa: (Running) La actividad está en primer plano. La pantalla del activity está visible y tiene la atención del usuario.
- Visible: (Paused) Este caso se suele dar cuando se abre una ventana emergente encima del activity. Sigue estando activa, pero deja de ser el foco de atención.
- Parada: (Stopped) El activity no es visible, se encuentra detenido.
- Destruída: (Destroyed) La actividad termina. Se hace llamada al método finish().

Para asignar a las activities los estados en los que se encuentran existen en Android métodos privados. Típicamente, el ciclo de vida de una actividad va desde onCreate() hasta onDestroy(). Cuando la actividad tiene estado visible parcialmente su ciclo sería onStart() hasta onStop(). Mientras que la actividad esté en primer plano siendo el foco de atención del usuario onResume() a onPause().

- onCreate(): se utiliza para la creación de la actividad. No tiene que ser obligatoriamente la creación de una interfaz de usuario, también se utiliza para la inicialización de bases de datos, por ejemplo.
- onStart(): La actividad va a ser visible al usuario.
- onResume(): es un método que se llama cuando la actividad va a ser mostrada al usuario. Aquí es donde colocaremos la música de inicio y las animaciones.
- onPause(): con este método mandamos pausar los eventos que creamos en el anterior, porque la actividad está próxima a pasar a segundo plano.
- onStop(): La actividad no es visible.
- onRestart(): tras haber llamado a onStop().

- onDestroy(): este método se llama previamente a que la actividad sea destruida por completo.



6. Ciclo de vida de una actividad

3.6.2. Abrir una actividad: los Intents

Normalmente, en cualquier aplicación pasamos de una pantalla a otra tocando algún botón. Es un gesto que tenemos tan interiorizado que no nos hemos parado a pensar como haríamos eso en programación. Para abrir una nueva actividad que nos de paso a otra pantalla de nuestra aplicación existen unos objetos llamados *Intent*. Podríamos decir que los componentes de una aplicación (las diferentes pantallas) están aisladas y la única manera que tiene de compartir información y conectarse es mediante los Intents.

Tipos de Intent:

- Explícito: Creamos un componente Intent y especificamos el nombre del componente de destino en el intento.
 1. Partimos de la base que tenemos dos activities: MainActivity.java y Activity2Main.java. Nos encontramos en la primera actividad y mediante un botón queremos pasar a la segunda.

```
Intent intentName = new Intent (MainActivity.this, Main2Activity.class);
startActivity(intentName);
```

Creamos un objeto tipo Intent. Tenemos que marcarle la referencia desde donde estamos (this o MainActivity.this) y a donde queremos ir (tenemos que referenciarlo .class y definir la activity en el AndroidManifest).

2. En este caso mandamos los datos del segundo activity al primero.

```
Intent intentName = new Intent(this, MainActivity.class);
```

```

intentName.putExtra("Name:", "Andrea"); //nombre: name | valor: Andrea
intentName.putExtra("edad", 22); //nombre: edad | valor: 22
startActivity(intentName);

```

Ahora queremos recuperar los datos de Name y Edad, creamos en MainActivity un objeto de tipo Bundle que nos permita recuperar los datos y un bucle que nos compruebe si hay dato.

```

Bundle extra = getIntent().getExtras();
If(extras !=null){
String name= extras.getString("name:");
Int edad= extras.getInt("edad");
}

```

- Implícito: Las intenciones implícitas especifican que una acción puede invocar a cualquier aplicación del dispositivo que sea capaz de realizarla. Resulta útil esta opción cuando nuestra aplicación no es capaz de realizar la acción en cuestión. A la hora de trabajar con este tipo de Intents debemos especificar la acción que queremos realizar y pasarle algún dato que necesite. Los datos que se especifican se pasan como tipo Uri.

```

Intent myIntent= new Intent(Intent.ACTION_VIEW, Uri.parse(http://www.instagram.com));
startActivity(intent);

```

3.7. Layouts

Podemos definir "Layout" como la forma que tenemos de acomodar los elementos que componen la interfaz de nuestra aplicación. Podemos decir que actúa como un contenedor de elementos siendo éste la referencia (raíz o root). Como hemos hablado anteriormente, las características de los layouts se modifican a través del código XML. Esto permite tener las características de la interfaz separadas del código Java, confiriéndole flexibilidad al programa.

Los layouts tienen una serie de propiedades que son comunes:

`android:layout_width="XX" y android:layout_height="XX"`.






- 1- Por tamaño asignado: "134dp" Tamaño fijo. Tenemos que poner especial atención con este modo, ya que la aplicación no se adapta al tamaño de la pantalla.
- 2- Adaptado al contenido: "wrap_content". El tamaño varía en función del contenido del objeto adaptándose a él.
- 3- Adaptado al padre: "match_parent". El widget se adapta al tamaño del padre que se asigne.

Otras propiedades para indicar posición:

- a. Posición relativa: `android:layout_above/below/alignLeft/alignRight ...`
- b. Posición relativa al layout padre:
`android:layout_alignParentLeft/alignParentRight/alignParentTop..`

- c. Referencia respecto de márgenes exteriores: `layout_marginXXX`
(Left,Right,Top,Bottom)
- d. Posición respecto de márgenes interiores: `android:padding/paddingXXX`

A continuación, trataremos los layouts más comunes son: Linear, Relative, Constraint y Grid.

-  **LinearLayout (vertical)**  **LinearLayout (vertical)** : el diseño lineal se caracteriza por organizar sus elementos en filas, que pueden ser de tipo horizontal o vertical:
`android:orientation="vertical"/"horizontal"`
-  **RelativeLayout** : el diseño relativo permite colocar los objetos en referencia de otro objeto o de ellos mismos. Por ejemplo, colocamos un objeto A con una serie de cotas y a continuación, colocamos otro objeto B con las cotas referenciadas al objeto A. De modo que si movemos el objeto A el B se moverá con él.
-  **ConstraintLayout** : desde la versión de Android 2.2, al crear un nuevo proyecto se crea por defecto un ConstraintLayout. Este tipo de layout permite establecer relaciones entre todos los elementos, incluso con la vista padre. Este layout incluye un modo que se llama Autoconnect, que crea restricciones automáticamente.
-  **GridLayout** : pertenece a ViewGroup. Este tipo de Layout alinea sus componentes en una cuadrícula. Se crea una cuadrícula:
`android:rowCount` y `android:columnCount`. A la hora de introducir un widget en una posición debemos especificar la fila y la columna:
`android:layout_row/column`. Podemos extender el objeto para que tenga mayor tamaño con `android:layout_rowSpan/columnSpan="nº de celdas que ocupa"` y activando la propiedad `layout_gravity="fill"`.

Existen otros tipos de Layouts como el `TableLayout` o `TableRow` que tienen bastantes conceptos en común con `GridLayout`, así que no los vamos a explicar en este capítulo, aunque sería interesante curiosear sobre qué tratan.

3.8. Widgets

A través de los Widgets configuramos la interfaz de usuario de nuestra aplicación. Se pueden definir como el componente visual a través del cual la aplicación interacciona con el usuario, proporcionando algún tipo de información a través de texto, casillas, eventos ...

A continuación, trataremos los componentes más relevantes para comenzar a trabajar en Android Studio.

3.8.1. TextView y EditText

El cuadro de texto puede tener carácter editable, que en ese caso sería `EditText`, y no editable, que sería `TextView`. Para asignar el texto al cuadro podemos hacerlo desde la parte derecha de la pantalla o bien podemos crearlo a partir del fichero XML con `android:text`. Respecto al tamaño que tiene el cuadro de texto, podemos aplicarle las mismas propiedades que a los layouts, puede ajustarse al contenido, al padre o tener un tamaño fijo.



7. Cuadro de texto

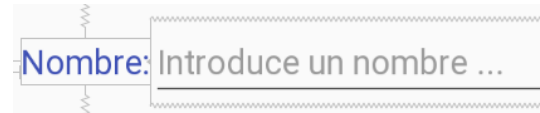
Algunas propiedades que podemos cambiar desde el fichero XML que pueden resultarnos realmente útiles son:

`android:typeface` – cambiar entre normal/sans/serif/monospace

`android:textColor` – Podemos cambiar el color del texto. Previamente debemos de registrar el color (`#RRGGBB`) con su identificador (`@color/nombreColor`) en el fichero XML para tener un acceso más rápido. También podemos utilizar el código RGB si es un color puntual.

`android:textSize` – Ajustar el tamaño del texto del cuadro en pixeles independientes de la densidad (dp).

`Android:hint` – Con hint nos referimos al texto que aparece por defecto en un `EditText` previamente a introducir información el usuario. En este ejemplo, el `TextView` nos indica que en el campo de alado debemos introducir el nombre, y en el `EditText` hay un hint que nos indica dónde debemos introducir el nombre.



8. Ejemplo de texto con hint

Nota: tenemos que tener en cuenta que Android por defecto utiliza las tipografías Droid Sans, Droid Serif y Droid Sans Mono. Es posible utilizar otra tipografía en nuestra aplicación, pero para ello deberemos cargar el archivo.ttf desde las herramientas.

Respecto a los métodos que podemos utilizar con estos campos, existen infinidad de métodos, aquí solo nombraremos los más relevantes para comenzar:

- `getText()` obtiene el texto que introducimos en `EditText` o bien el texto residente en el `textView`.
- `setText()` introduce la cadena de caracteres en el cuadro de texto.
- `clear()` limpia el campo.
- `moveCursorToVisibleOffset()` permite desplazar el texto lateralmente cuando excede el tamaño del campo e impide su visualización completa.

3.8.2. Button

`Button` es una subclase de `TextView`. Los botones realizan una acción cuando el usuario pulsa sobre ellos. Por ser subclase de `TextView`, tiene una serie propiedades en común. Todo lo referido con anterioridad al fichero XML del `TextView` es aplicable a la etiqueta del botón. Podemos cambiar la forma y el color del botón modificando sus atributos. Es interesante comentar un par de método que son bastante útiles:

`android:onClick` – hace referencia al método que se llevará a cabo cuando se presione el botón.

`android:enabled` – “true” el botón está activo(por defecto), “false” botón inactivo.

Los botones pueden tener una imagen de fondo, en lugar del color plano que sale por defecto. Si deseamos poner una imagen en concreto, debemos introducir el fichero (preferiblemente en *.png) en la carpeta `drawable`. Debemos cambiar la propiedad



9. Botones

android:background y escoger nuestra imagen. También existe una opción que permite coexistir el texto, el color de fondo y el icono de manera ordenada dentro del botón. Debemos ordenar el icono a través de las propiedades android:drawableLeft/right/Bottom...

¿Cómo comunicamos al botón que debe realizar una acción? Pues bien, existen diferentes maneras de hacerle saber al botón que debe realizar un acción. Lo primero de todo, debemos definir un método en Java que reciba un View y devuelva void. Por ejemplo:

```
public void onClick(View v1){  
    myTextView.setText("Ha hecho click");}
```

Después debemos implementar un Listener. Podemos hacerlo mediante diferentes maneras:

1. Implementando desde Java un evento setOnClickListener indicando el botón que debe realizar la acción. Android detecta el método y crea automáticamente un método onClick que contendrá la acción a realizar:

```
myButton.setOnClickListener(  
    new View.OnClickListener() {  
        @Override  
        public void onClick(View view) {  
            myTextView.setText("Hola mundo!");  
        }  
    }  
);
```

2. Desde la interfaz de usuario: nos situamos en la parte derecha de la ventana Design. Entre los atributos de Button encontraremos onClick y en el desplegable se encontrará el nombre de nuestro método definido previamente. Con esta manera de asignar la acción al botón no es necesario crear un Listener en el archivo Java.
3. Desde el XML: básicamente hace lo mismo que desde la interfaz, pero a través del código. La propiedad que debemos activar es android:onClick="..." y entre las comillas debemos poner el nombre del método, que en nuestro caso es "onClick".

3.8.3. Radio Button y RadioGroup

Los RadioButton que incluyamos en nuestra aplicación deben estar incluidos dentro de un contenedor de tipo RadioGroup. Debemos arrastrar a la interfaz de nuestra aplicación un RadioGroup y dentro soltar tantos RadioButton como queramos. Todos los RadioButton que estén dentro de la misma agrupación se excluirán mutuamente.



Desde el código Java podemos modificar las propiedades del RadioGroup utilizando los siguientes métodos:

void check(int id): activa el RadioButton con el identificador "id".

void clearCheck(): Limpia la selección dejando todos los botones desactivados.

int getCheckedRadioButtonId(): recoge el "id" del botón seleccionado.

`void toggle()`: cambia el estado del botón.

`void onCheckedChanged(RadioGroup, int checkedId)`: La llamada se activa cuando cambia el estado del botón.

Para utilizar el `RadioGroup` es necesario implementar un `Listener`:

- a. Podemos utilizar `android:onClick` desde el XML. Le asignamos a cada `RadioButton` la acción que debe realizar en caso de cambiar su estado. Este tipo de llamada se realiza cuando todos los botones realizarán la misma acción, es decir, que se puede realizar desde un método general.
- b. el `Listener`: `setOnCheckedChangeListener()`. Este `Listener` registra cuando hay un cambio en el `RadioGroup`.

```
radiogroup.setOnCheckedChangeListener(new RadioGroup.OnCheckedChangeListener()
{
    @Override
    public void onCheckedChanged(RadioGroup group, int checkedId) });
}
```

Con el método `onCheckedChanged(RadioGroup group, int CheckedId)`: `checkedId` registra el `Id` del botón que se ha pulsado. Aquí podemos hacer uso de los condicionales, ya que en función del `Id` podemos realizar diferentes opciones.

A raíz de este ejemplo vamos a explicar otro concepto de bastante utilidad:

3.8.4. Los Toasts

Los `Toasts` se puede definir como mensajes que se muestran en pantalla durante unos segundos y después desaparecen.

- No interfieren en las operaciones que está realizando la operación.
- Son a título informativo, no requieren ninguna respuesta por parte del usuario.

Para utilizar un `Toast`, primero debemos declarar un objeto de esa misma clase. Después le añadimos una serie de características que nos indican su definición como, por ejemplo:

```
Toast myToast;

myToast.makeText(Context context, CharSequence
text, int duration);

[Context : getContext(), " Texto que aparecerá",
LENGHT_LONG/SHORT]

myToast.setGravity(int gravity, int xOffset,int yOffset);

myToast.show(); [muestra el toast en pantalla]
```

¿Qué asignatura prefieres?

Matemáticas

Física

Expresión Gráfica

Dibujo Artístico

Matemáticas

10. Ejemplo de Toast

3.8.5. ProgressDialog y ProgressBar

Siguiendo con los mensajes que nos permiten mostrar notificaciones al usuario, tenemos el `ProgressDialog`. Este tipo de cuadros de diálogo nos permiten indicar al usuario que se está ejecutando una operación, normalmente en segundo plano, así como ver el estado de su progreso.

Debemos introducir en la interfaz un componente, como por ejemplo un botón, que lo active. Creamos dentro del método onClick un objeto de tipo ProgressDialog/Bar y le asignamos unas características: el mensaje que mostrará, el tipo de barra de progreso, así como el mínimo inicial. En el caso del ProgressDialog debemos activarlo usando la función show().

Llegamos al momento donde tenemos que hablar de otro concepto importante:

3.8.6. Los Threads

Para comenzar a hablar de los Threads tenemos que definir el concepto Hilo. Un hilo es una parte del código que se encarga de realizar alguna acción a la vez que se va ejecutando otra. Se utiliza para realizar actividades simultáneas.

Los Threads son objetos que permiten ejecutar operaciones en segundo plano, en este caso, cargar nuestro ProgressDialog en un segundo plano.

Para explicarlo de una manera más sencilla, podemos seguir el siguiente esquema para lanzar una actividad en segundo plano:

```
método onClick(){
    Thread(new Runnable(){
        public void run(){
            [Programamos las acciones que se realizarán en segundo plano]
        }
    })
}
```

En definitiva, creamos un constructor de la clase Thread que recibe como parámetro un objeto de tipo Runnable. En el interior del método llamado run() es donde programamos nuestra actividad en segundo plano. Finalmente, debemos asignarle a al Thread la acción de start(); para que comience a funcionar.

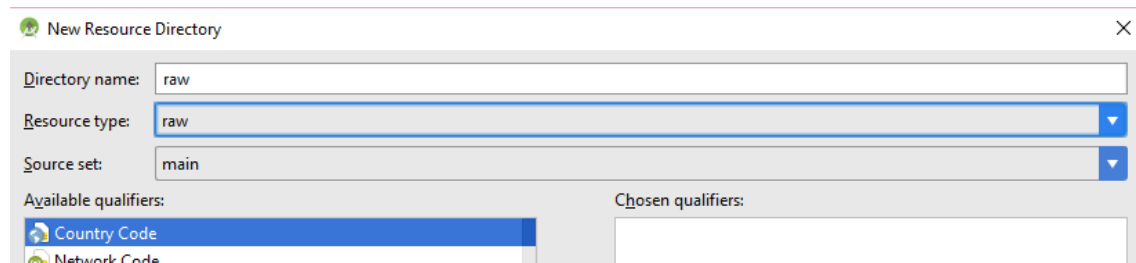
3.8.7. Reproductor de sonido

Como habrás podido observar, algunas aplicaciones incluyen sonido de inicio, efecto al pulsar un botón, reproductor de música ... Para reproducir música en nuestra aplicación tenemos diferentes clases: MediaPlayer y SoundPool.

1. MediaPlayer: esta clase la implementaremos para reproducir archivos de sonido largos como, por ejemplo, canciones completas o música de fondo para la app.

Los formatos que soporta el programa son: WAV, AAC, MP3, WMA, AMR, OGG y MIDI.

Para utilizar este componente, es necesario crear una nueva carpeta donde incluiremos los archivos de música. Dentro del directorio res, abrimos el desplegable de opciones con el botón derecho y hacemos clic sobre: New/Android Resource Directory. A continuación, se abre un cuadro de opciones donde, por defecto aparece Resource type: values. Debemos cambiar el tipo a: raw.



11. Carpeta para incluir archivos de sonido

Una vez realizado este paso, ya tenemos la carpeta donde almacenar los archivos de sonido. Copiamos de nuestros Documentos el archivo deseado y pulsamos botón derecho sobre la carpeta raw: paste.

Respecto al código Java, debemos crear un objeto de la clase MediaPlayer. Utilizamos el método: `myMediaPlayer=MediaPlayer.create(context, ruta_archivo_sonido)`. Otros métodos básicos que podemos utilizar son: `play()`, `pause()`, `reset()`, `isPlaying()`: devuelve true/false.

2. SoundPool: Nos permite reproducir archivos de sonido de tamaño máximo 1Mb. Por sus características se utiliza para reproducir sonidos cortos, tales como los efectos de los botones o recursos de audio. Los archivos de sonido los almacenamos en la misma carpeta que los de MediaPlayer.

En cuanto al código que tenemos que programar en Java, primero debemos crear el objeto de tipo SoundPool así como una variable de tipo int donde almacenaremos la canción.

1. Creamos un nuevo objeto SoundPool:

```
MySPool= new SoundPool(int maxStreams=1, int StreamType=
AudioManager.STREAM_MUSIC,int srcQuality =1);
```

- maxStreams: número de veces que se repite el sonido.
- streamType: tipo de flujo.
- srcQuality: indicador de calidad.

2. Igualamos la variable entera a la estructura de carga, que es un número entero con un flujo de información. Para referenciar el flujo de información utilizamos la siguiente estructura:

```
int cancion = MySPool.load (Context context, int resId, int priority);
```

Context: this

int resId: aquí va la ruta a la canción: R.raw.mi_cancion

int priority: normalmente será 1

3. Activamos el SoundPool utilizando la función `mySPool.play()`. Podemos introducir esta función, por ejemplo, cuando implementamos la

función `onClick` de un botón, para que al hacer clic sobre él tenga efecto de sonido. A continuación, explicamos la estructura del método por ser algo extenso:

```
mySPool.play(soundID, leftVolume, rightVolume, priority, loop, rate);
```

- `soundID`: nombre de la variable de almacenamiento de flujo de información.
- `leftVolume`: volumen de la salida de audio izquierda.
- `rightVolume`: volumen de la salida de audio derecha.
- `priority`: 1
- `loop`: Podemos repetir el sonido asignando a este parámetro un valor diferente de cero. Si `loop=-1` el sonido se repite de forma ininterrumpida hasta que se produzca la llamada a la función `stop()`. Si `loop` toma el valor de cualquier otro número entero positivo, el sonido se repetirá $n+1$ veces.

4. Desactivamos `SoundPool` en caso de que hayamos implementado un bucle infinito: `mySPool.stop(cancion)`; Debemos referenciar que flujo de información queremos parar porque puede que haya diferentes activos en el mismo momento.

3.9. Material Design

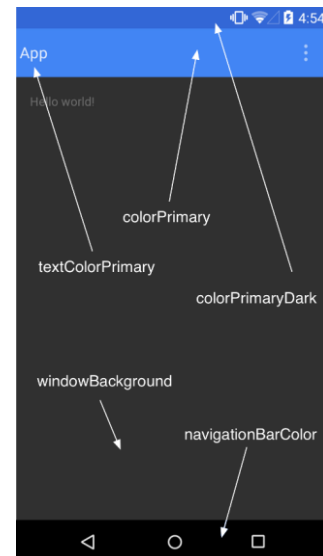
Material Design se concibe como la guía que se utiliza para el desarrollo de aplicaciones basadas en el diseño visual, de movimientos e interacciones entre los diferentes objetos. Los objetos materiales están colocados en un espacio y en un tiempo concretos. Se crea sensación de profundidad a través de las sombras y los colores, pues se intenta acercar la realidad al mundo virtual. Por ello, Material Design pretende ser fiel a las leyes de la física, es decir, intenta crear animaciones lógicas dejando de lado otras cuyos objetos se atraviesan o traspasan.

El máximo responsable de este “movimiento” es el chileno Matías Duarte. Antes de introducir los diferentes componentes que permiten crear aplicaciones basadas en Material Design, trataremos unos conceptos generales:

Material Design es aplicable a partir de un API 21, es decir, a partir de Android 5.0.

El tema Material tiene tres definiciones: `Theme.Material` / `Material.Light` / `Material.Light.DarkActionBar`. Este tema permite definir una paleta de colores en función de los colores de la aplicación. Los colores previamente definidos son:

- `colorPrimary`
- `colorPrimaryDark` – Color de la barra de estado. Ésta debe estar claramente delimitada, bien sea con un color plano o con un degradado para permitir la correcta visualización de los iconos.
- `navigationBarColor` – Aunque existen variantes, será principalmente de color negro.
- `windowBackground`
- `textColorPrimary`



12. Nomenclatura

Los elementos de una aplicación deben estar ordenados de manera clara y sencilla: la tipografía debe ser clara, los colores e imágenes que se utilicen serán llamativos para que el usuario no pierda en ningún momento el foco así como la sensación de una aplicación jerarquizada y ordenada. El movimiento es el mejor aliado para guiar al usuario a través de la aplicación, sin que éste pierda el foco. En este sentido, la iluminación de la aplicación debe ser realista, pues ha de proporcionar la información suficiente para que el usuario pueda suponer o deducir como actuarán los botones. Mediante las sombras se crea la jerarquía. Para realizar una aplicación más accesible se intentará reemplazar las palabras por iconos cuyo significado sea claro y conciso, dándole así una mayor accesibilidad, ya que se elimina la barrera del idioma. Los menús se dispondrán siempre que sea posible en horizontal y las imágenes deben tener un contenido atractivo.

En cuanto a los iconos, debemos utilizar el mayor tamaño posible (XXHDPI o XXXHDPI) en las imágenes de los mismos para evitar que se pixelen cuando cambiemos de plataforma o de tamaño de pantalla. Siempre teniendo en cuenta que es mejor trabajar con imágenes vectoriales, pues así evitamos el problema del pixelado.

Android pone a disposición pública un generador de iconos, que se basa en los principios de Material Design, llamado Android Assets Studio. La ventaja que tiene utilizar este generador es que al crear un icono se crea automáticamente un paquete de iconos con todos los tamaños necesarios.



13. Iconos

3.9.1. AppBars

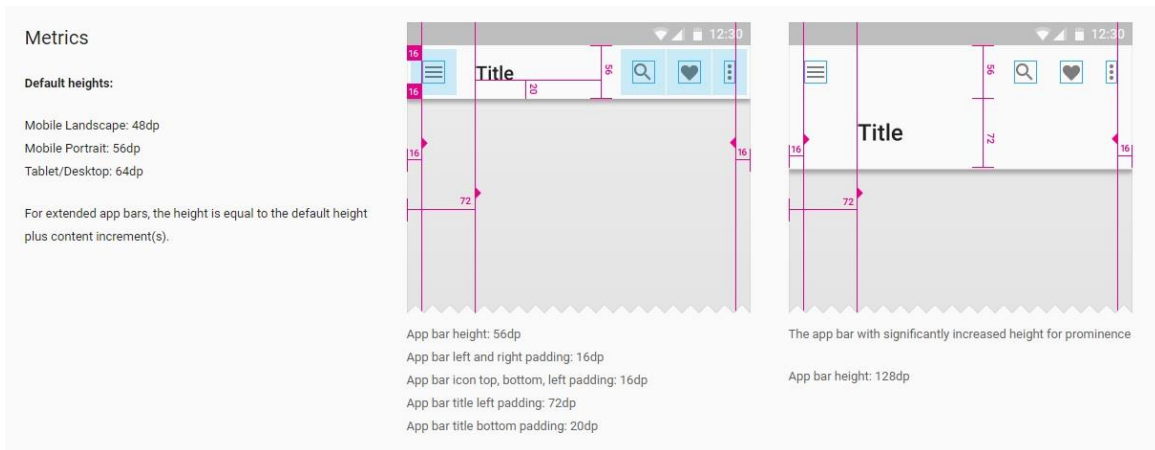
Comúnmente conocidas como Action Bar o Barras de aplicación.



1. Nav Icon: Se utiliza para abrir el Navigation Drawer que contiene normalmente un menú superior a la pantalla de la aplicación que se está mostrando. También puede ser una fecha que indica que retrocedemos en la jerarquía.
2. Title: Contiene el nombre de la aplicación que normalmente es el nombre del fichero. Se puede cambiar desde las propiedades del archivo.xml en `android:label="title"`
3. Action Icons: representan las acciones principales de la aplicación, o las que mas se suelen usar. El resto de acciones que se conciben como secundarias están en Overflow.
4. Overflow options menú: Aquí es donde residen las acciones que no son tan importantes como para situarse en la appBar.

La appBar se puede dividir en dos zonas, siendo los iconos de la parte derecha relativos a la aplicación en particular. Como características comunes podemos señalar que tienen que ser todos del mismo color, pudiendo destacar el título en otro color para hacerlo más visible.

A continuación, se muestra una foto con las medidas estándar de la appBar:



14. Estándar de medidas de appBar

3.9.2. Bottom Sheets

Los Bottom Sheets no son más que hojas desplegables que aparecen como resultado de una acción. Se puede arrastrar de forma vertical hacia la parte superior de la pantalla para mostrar más contenido.

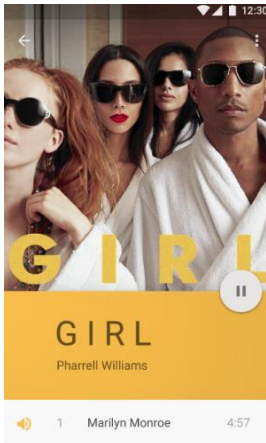
Los Bottom Sheets tienen cinco estados:

- **STATE_COLLAPSED:** Es el estado mediante el cual se hace presente la hoja y el usuario se da cuenta que existe contenido extra. Para ver ese contenido debe realizar otra acción como deslizar o arrastrar el Bottom Sheet. Coloquialmente podemos hacer referencia a este estado como “posición de reposo” o “resting position”.
- **STATE_EXPANDED:** La hoja se encuentra al máximo de su tamaño y no está en posición de Dragging ni Settling,
- **STATE_DRAGGING:** Estado en que se encuentra el Bottom Sheet mientras el usuario está desplazándolo hacia arriba o abajo.
- **STATE_SETTLING:** La hoja se ancla a una altura determinada después del gesto de arrastrar. Esta altura puede ser: ojeada(collapsed), expandida(expanded) o hidden, es decir, oculta.
- **STATE_HIDDEN:** Bottom Sheet se encuentra a una altura igual a cero. No es visible.

Tipos:

1. **Persist Bottom Sheets:** Se encuentran integradas en la aplicación para mostrar el contenido de apoyo. La elevación es la misma que la de app. Algunas de sus funciones son: presentar el contenido de la aplicación, ser una vista de apoyo a la principal, muestran un resumen del contenido, o desarrollan alguna actividad complementaria como reproducir música o mostrar un mapa.
2. **Modal Bottom Sheets:** Se conocen como diálogos auxiliares o menús que permiten enlazar nuestra aplicación con funciones de otras aplicaciones, tales como mandar mensajes, buscar en Google, abrir GoogleMaps...

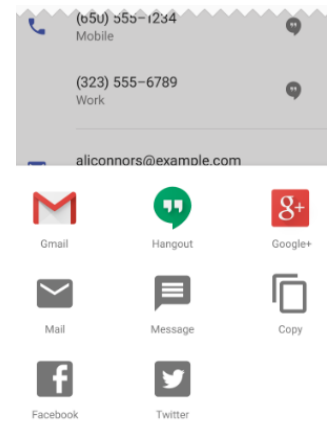
Aparece sobre la pantalla principal, mientras que ésta se desactiva y atenúa tornando a un color grisáceo. Este efecto hace que el foco pase a ser la Modal Bottom Sheet. El menú contextual se puede mostrar en forma de cuadrícula, o bien como una lista. El menú extendido al máximo debe guardar con la parte superior una distancia mínima de 8dp.



15. Persist Bottom Sheet



16. Bottom Sheet



17. Modal Bottom Sheet

3.9.3. Coordinated Behaviors

CoordinatorLayout es un contenedor de uso general que permite implementar comportamientos coordinados como los que hemos visto anteriormente, por ejemplo, desactivar la pantalla mientras se muestra una Modal Bottom Sheet.

Hay dos tipos de casos:

- 1- Diseño de contenido de nivel superior, siendo la raíz de las vistas.
- 2- Contenedor de una vistas general específica de actividades secundarias.

3.9.4. Bottom Navigation

Bottom Navigation crea una forma sencilla de navegar entre los diferentes niveles de una aplicación. Para hacernos una idea, un ejemplo bastante conocido es Instagram. Este tipo de objeto puede usarse cuando se tienen un mínimo de tres pantallas y un máximo de cinco.

Características:

- Los botones tienen acceso directo a las localizaciones.
- Cuando se pulsa el botón, éste se debe poner del color primario de la aplicación. Si ya está coloreado, se usará el color blanco o negro.
- Solo se debe utilizar un único color, para resaltar el foco.
- Número de niveles:
 - Tres niveles: se usa icon + label (corta)
 - Cuatro/cinco niveles: se usan iconos y se muestra opcionalmente la etiqueta del que está activo.
- La barra de navegación permite la movilidad desde el final de un nivel hasta el principio del otro, o de ese mismo, volviendo a pulsar el botón correspondiente. Este movimiento debemos de mostrárselo al usuario de forma clara.
- Otro gesto que debemos dejar claro es que al hacer Scroll hacia abajo la barra desaparece, y por el contrario, si lo hacemos hacia arriba, se muestra apareciendo de abajo a arriba.



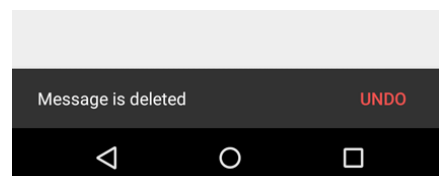
18. Ejemplo real de Bottom Navigation (Instagram)

No se debe utilizar Bottom Navigation para:

- Vistas de una sola tarea.
- Vistas de preferencias de usuario o configuraciones.
- El botón de retroceder no debe estar entre los botones de la barra de navegación.

3.9.5. SnackBars

Quizás por el nombre no seamos capaces de reconocer una Snackbar, pero las tenemos presentes en prácticamente todas las aplicaciones. Este tipo de barras proporcionan un rápido feedback al usuario, haciendo su aparición en la parte inferior de la pantalla.



19. Snackbar

Características:

- Permiten realizar una acción de una manera rápida, que normalmente es rehacer o deshacer una acción previa.
- Aparecen de manera automática, y permanecen en pantalla según la duración asignada LENGTH_INDEFINITE/LONG/SHORT. También pueden desaparecer si el usuario los arrastra hacia abajo o si interactúa con él.

Hemos de mencionar, que los Snackbar trabajan mejor en un CoordinatorLayout, ya que permiten la acción swip-to-dismiss.

- Los Snackbars no contienen iconos.
- Solo se puede mostrar un SnackBar por pantalla al mismo tiempo.
- No se deben colocar acciones tales como “descartar” o “cancelar”.
- Para asignarle una acción se utiliza el método `setAction()`

Especificaciones:

- Action Button: Roboto Medium 14sp, en mayúsculas. (donde situaremos la acción)
- Altura: línea única(48dp) y multilínea (80dp)
- Nunca pueden aparecer por encima del Floating Action Button.

3.9.6. Text Fields

TextInputLayout nos permite crear campos de texto siguiendo los principios de Material Design. Podemos usarlo de manera conjunta con `TextInputEditText` ya que nos facilitará la tarea. Utilizamos estos objetos para validar información de entrada a la aplicación, nos permiten corregir errores ortográficos, autocompletar palabras mediante la predicción, así como proporcionar sugerencias de escritura.

Características:

- 1- Deben ser fácilmente identificables para los usuarios de la aplicación, es decir, que entiendan que en ese espacio pueden introducir información.
- 2- Debe ser lo suficientemente localizable como para identificarlo entre otros elementos.
- 3- Debe tener buena legibilidad, así como fácil comprensión de su estado:
Habilitado/deshabilitado, vacío/completo, válido/completo.
- 4- Puede contener texto de asistencia.

Apariencia del TextField:

1. Label: La etiqueta indica el tipo de texto que requiere el campo. Puede encontrarse alineado con el texto(resting) o bien situado encima de él (floating).
2. Línea de entrada: Indica dónde se introduce el texto. Cuando el campo se encuentre activo la línea se volverá más gruesa.
3. Cursor: como en otro tipo de entrada de datos, el cursor indica dónde está situado el usuario.
4. Entrada de texto: Se debe colocar en mayúscula la primera letra.
5. Hint: ya hemos visto el Hint en otros capítulos pero, para resumir, el Hint nos permite tener una idea del tipo de texto que se espera introducir en el campo y cuando éste se encuentra activo, desaparece.
6. Texto de ayuda: texto opcional que aparece en la parte inferior, debajo de la línea de entrada. Debe colocarse en una sola línea y suele contener alguna especificación sobre el texto a introducir.
7. Mensaje de error: aparece cuando el texto introducido no es el correcto. El usuario visualiza el mensaje de error junto con otro que propone una solución o un aviso. Mensaje: Error: “mensaje de la app” ó ⓘ “mensaje de la app”

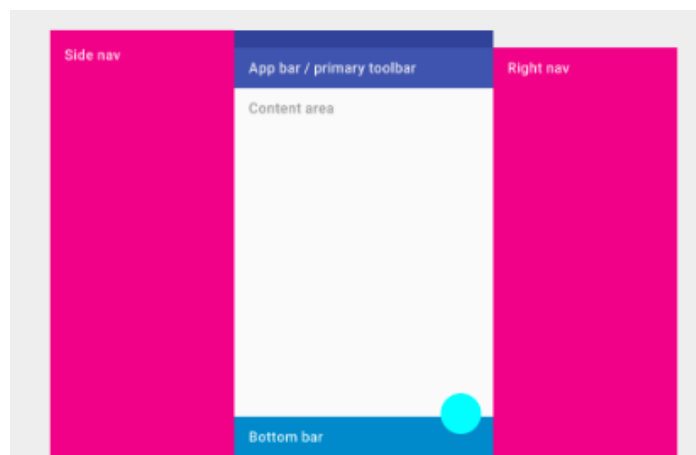
8. Contador de caracteres: existen ciertos campos que requieren un número máximo de caracteres. (caracteres disponibles/caracteres totales)
9. Icono: en ocasiones aparece delante del texto un icono, por ejemplo, de correo electrónico.
10. VoiceInput: para introducir texto mediante voz.
11. Dropdown Icon: indica que el campo tiene un desplegable con opciones.
12. Clear button (X): botón para limpiar el campo.

Interacciones del usuario:

- Hover: cuando el cursor pasa por encima del campo, la línea adquiere un grosor mayor.
 - Press: el label se desliza hacia arriba y pasa de estar en modo resting a floating.
- A. Text Área: son objetos más grandes que los TextField de una única línea. Se desplazan de manera vertical para permitir visualizar el texto completo. Tiene una línea de contorno que comparte grosor y color con la línea de entrada de texto.
 - B. Input Types: pueden tener formatos diferentes:
 - a. Grupos de caracteres: Se agrupan los caracteres y se introducen caracteres especiales entre grupos, como los prefijos o el IBAN de una cuenta bancaria.
 - b. Prefijos y sufijos: se añaden al principio y/o al final de la línea de entrada de texto, por ejemplo, para indicar unidades.
 - c. Contraseñas: las contraseñas aparecen ocultas por defecto. En su lugar aparecen puntos, que son conocidos como midline elipses. Alado de la contraseña aparece un icono de visualización que permite saber si la contraseña está visible o no pulsando sobre él.
 - d. Menús and Pickers: desplegables que permiten escoger una opción para rellenar el campo de texto.

3.9.7. Collapsing Toolbars

El Collapsing Toolbar Layout es un ViewGroup que presenta numerosas características e interacciones entre sus componentes:



20. Esquema de un Collapsing Toolbar

AppBarLayout,CoordinatorLayout,Toolbar y una lista de contenido desplazable, como por ejemplo, un RecyclerView.

- Los elementos que componen que componen la AppBar son los mismos que hemos visto en el punto de AppBar.
- SystemBars: Contiene los iconos de las notificaciones, así como los iconos de información de batería, cobertura, sonido ...
 - a. Full-Screen: Este modo se utiliza para ver vídeos y fotografías ya que proporciona una mejor experiencia para el usuario, minimizando las distracciones y protegiéndolo de salir de forma accidental del contenido que están reproduciendo.

Tipos:

- 1- Lean-back: Ideal para ver videos. Las barras desaparecen a los pocos segundos de comenzar la reproducción. Para volver a verlas debemos de tocar sobre cualquier parte de la pantalla.
 - 2- Inmersive: Es la mejor opción cuando los usuarios están interaccionando con la pantalla. Los controles se muestran y ocultan deslizando.
 - 3- Lights- out: La ActionBar y StatusBar se desvanecen a los pocos segundos de la inactividad. Por el contrario, la NavigationBar sigue disponible, pero aparece atenuada. La Android Navigation Bar suele aparecer en blanco o negro(modos Dark o Light) pero existen variantes dependiendo de la acción que esté desarrollando la aplicación: translúcida, translúcida con imagen de fondo o transparente elevada sobre fondo.
- Slide Nav Bars: Son las barras de navegación laterales. Estas barras pueden fijarse de forma permanente o bien, permanecer flotantes de forma temporal sobre las capas. Se encuentran ubicadas a derecha o izquierda de la pantalla.

Contenido derecha: Aloja contenido secundario de la pantalla principal. El tamaño de esta capa cubre la pantalla entera.

Contenido izquierda: Se suele utilizar para la navegación. El tamaño de esta capa es $Width = Screen\ Size - 56dp$ siendo el máximo 320 dp.

Estructura:

- 1- Nav Drawers: Se superponen al lienzo. Se suele utilizar cuando no hay espacio suficiente.
 - 2- Pinned Nav: Están situados a lo largo o debajo del lienzo del contenido.
- White Frames: Proporcionan una gran variedad de estructuras utilizando las superficies, las capas y las sombras.

3.9.8. Navigation Views

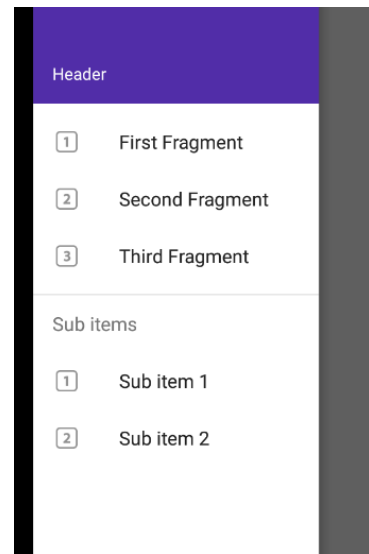
Se puede definir Navigation View como una forma rápida y sencilla de visualizar un menú de contenido. Estos objetos suelen utilizarse juntamente con un DrawerLayout, que permite un funcionamiento de alto rendimiento. Los Navigation Drawes son diálogos que aparecen elevados desde la parte izquierda de la pantalla. Se utilizan para mostrar links de navegación de la aplicación. En las últimas versiones de Android, el icon nav ha desaparecido, de manera que solo se muestra el menú deslizando desde la parte izquierda de la pantalla hasta el centro.

Para construir un Navigation View necesitamos:

- Un DrawerLayout con dos hijos: el contenido principal y el Navigation View.

- Un layout para el Header (opcional). El Header es el diseño del encabezado. Comúnmente aparece el correo electrónico y una foto de usuario. Para implementarlo debemos crear un archivo en XML llamado nav_header.xml. Aquí es donde albergaremos los datos necesarios para configurar el header.

- Menú para implementar las opciones.



21. Esquema de Nav Views

3.9.9. Floating Action Button

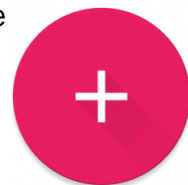
Quizás este será el objeto que más familiar nos resulte, ya que estamos constantemente viéndolo en las pantallas de las aplicaciones. El Floating Action Button (FAB) muestra la acción primaria de la pantalla de la aplicación.

Es un icono circular que se encuentra elevado por encima del contenido de la página. Puede transformarse en otros botones o elementos de la pantalla. Además, tiene la capacidad de movilidad, es decir, se adapta al movimiento de otros objetos de la página, de manera que, permanece siempre visible. Por ejemplo, si en una determinada acción en la pantalla sale una Snackbar, según la colocación del FAB ésta lo tapanía. El FAB se desplaza hacia arriba como empujado por la Snackbar. Por el contrario, si se abre una Bottom Sheet el FAB desaparecerá.

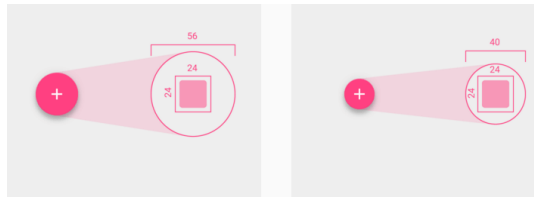
Sólo se permite un FAB por pantalla.

Propiedades:

- Imagen: se utiliza el comando android: src ="@ubic_foto" para cambiar el icono del interior.
En JAVA utilizamos el método setImageDrawable.
- Tamaño: utilizamos app:fabSize y en JAVA el método setSize.
Típicamente podemos hablar de dos tamaños para el FAB:
 - Default 56 dp. Se utiliza para acciones generales.
 - Mini: 24 dp. Se utiliza para seguir el patrón general.



22. Apariencia del Floating Action Button

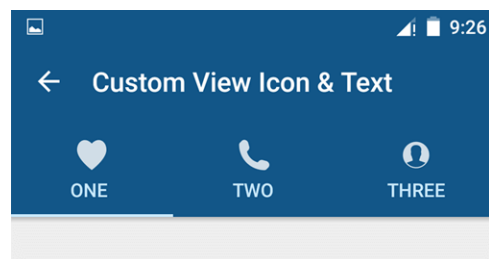


23. Tamaños estándares del FAB

- Color: Por lo general, se le aplica al FAB el colorAccent de la aplicación. Para cambiar el color utilizamos android:backgroundTint y en JAVA setBackgroundTintList
- Elevation: Utilizamos android: elevation y en JAVA setCompatElevation.
- Los comportamientos de movimiento del FAB son: Transformación, Lanzamiento y Anclaje de transferencia.
- Suelen representar acciones positivas. No se utilizará un FAB para errores o acciones que deberían estar en la barra de herramientas.
- En caso de cambiar de acción de una pantalla a otra debe desaparecer y aparecer. En caso de que haya varias pestañas en una pantalla no debe moverse con ellas al desplazarse.
- Transiciones:
 - Trigger: Ondulación táctil. Se expande desde el centro hacia fuera reflejando la fuerza del usuario para crear movimiento en la página.
 - Toolbar: El FAB se convierte en una barra de herramientas al tocarlo. Se sitúa a la misma altura a la cual se encontraba. Desaparece cuando se produce un desplazamiento de pantalla y vuelve a convertirse en botón.
 - Marcación rápida: Speed Dial. Sobre el FAB aparecen otros iconos de acciones principales, siendo de tamaño mini. Si volvemos a marcar el FAB, se esconden.
No contendrá acciones de desbordamiento, porque para ello existe el botón de desbordamiento en la barra de herramientas.
- Full-Screen: al tocar el FAB, éste se puede convertir en una hoja nueva ocupando toda la pantalla. No existe un movimiento para deshacer la transición de forma, de modo que el FAB se expande para crear la hoja, y luego se retrocede para recuperar la forma original. No existe una inversión.

3.9.10.Tab Layout

Tab se traduce como lengüeta, pestaña. Un Tab Layout es un layout horizontal provisto de pestañas para crear páginas. Este tipo de diseño de páginas crea interacciones entre Scroll, gestos de deslizar, selección de pestañas animaciones y alineación de objetos.



Características:

- 1- Debe de tener solo una única fila de pestañas y éstas deben estar agrupadas jerárquicamente.
- 2- No se deben incluir pestañas dentro de una pestaña.
- 3- Movimiento: fijo o desplazable.

- 4- El número de pestañas depende de varios factores: tamaño, número, y el tamaño de cada pestaña.
- 5- Todas las pestañas deben tener el mismo tamaño de texto.
- 6- El ancho de la pestaña se fija a partir del texto más largo.

El contenido de cada página se debe alojar en un ViewPager (container). En las etiquetas se puede utilizar: texto(label), iconos o texto + iconos. No se puede poner texto alternado con iconos.

Tipos:

- a. Pestañas fijas: Permiten cambiar rápidamente entre pestañas. Para navegar entre pestañas basta con tocar sobre ellas o arrastrar la pantalla.
- b. Pestañas desplazables: Cuando los usuarios no necesitan comparar las etiquetas de las pestañas y existen más pestañas de las que puede albergar el ancho de la pantalla. Para visualizarlas se debe deslizar el área de las pestañas hacia ambos lados.

4. Aplicación

4.1. Análisis de las principales aplicaciones para aprendizaje de idiomas

4.1.1. Duolingo

Duolingo se define como un proyecto social cuyo principal objetivo es el aprendizaje de idiomas. Se encuentra disponible de manera gratuita en las diferentes plataformas y existen diferentes cursos para hispanohablantes como, por ejemplo, inglés, alemán, italiano, portugués ...

La aplicación se encuentra disponible en diferentes plataformas: iOS, Android, Windows Phone y Chrome. El icono que utiliza Duolingo varía de unas plataformas a otras, siguiendo el diseño del sistema operativo, el primer caso es para Android y el segundo para iOS.



24. Iconos de Duolingo

La metodología de trabajo de esta aplicación se basa en que la mayor parte de las oraciones se pueden comprender sin necesidad de saber todo el vocabulario, simplemente con ayuda de imágenes y el corrector que nos indica los errores.

A continuación, se analiza la interfaz de la pantalla principal de la aplicación, identificando los principales elementos según lo estudiado en el capítulo anterior.

Android Status Bar de color negro, igual que la **Barra de Navegación**.

AppBar Contiene:
-Navigation Drawer, que despliega un menú principal de los cursos que estamos realizando.
-Título: del curso actual.
- Action Icons: meta diaria y fluidez de idioma.
- Icono de

Bottom Nav. Contiene los accesos de las principales pantallas de la app:
Aprender, Perfil, Clubs y Tienda.
Solo muestra los iconos, excepto del que está seleccionado que muestra la etiqueta también.

Floating Action Button Contiene la acción principal de la aplicación:
Entrenar para alcanzar la meta diaria.

4.1.2. TinyCards

Desarrollada por los creadores de Duolingo, TinyCards se presenta como una aplicación destinada a memorizar vocabulario relacionado con algún idioma o con otra materia como, por ejemplo, matemáticas o química. Lo interesante de esta aplicación es la metodología de trabajo, que implica de manera activa y lúdica al usuario.



25. Icono de TinyCards

En este caso el icono que utiliza, tanto en Chrome como en las versiones móviles Android e iOS, es el mismo.

Análisis de la interfaz de TinyCards:

Tab Layout compone el cuerpo de la aplicación donde se alojan las principales páginas:
Home, Search, Create Cards y Profile.

El desplazamiento sólo se produce al tocar sobre el botón correspondiente. No permite deslizar la pantalla.

Android Status Bar del color principal de la app, la **Barra de Navegación** de color negro.

No tiene AppBar con las acciones principales ni menú desplegable oculto en la parte izquierda.

Al tocar el botón de Search se activa un Toolbar con una barra de búsqueda que reemplaza a la barra de pestañas.

Diferentes pantallas de la aplicación donde se muestra el sistema de las cartas o tarjetas. Por una cara se muestra la palabra en el idioma de inicio, y por el otro la traducción. Tras practicar un número determinado de veces, la aplicación te pide que escribas la palabra en un **Text Field**.

4.1.3. Busuu

Busuu es otra de las aplicaciones más conocidas a nivel de aprendizaje de idiomas. Ofrece la posibilidad de aprender hasta 12 lenguas diferentes.

En esta aplicación, cuyo nombre procede del lenguaje hablado en Camerún: el busuu, los usuarios practican el idioma escogido mediante ejercicios para mejorar la comprensión oral y escrita, así como la comprensión escrita. Existen dos puntos de partida, bien como principiante o mediante una prueba de nivel, para situar al usuario dentro de un nivel de la aplicación (B1, B2, A1 y A2). La metodología de aprendizaje se lleva a cabo mediante ejercicios de gramática y vocabulario reforzados con audio e imágenes. Tiene un cierto parecido a TinyCards, ya que utiliza la metodología de las tarjetas para marcar las partes.



26. Iconos

Es posible utilizar la aplicación sin implicarse en la red social que propone el contacto y ayuda con otros usuarios mediante la corrección de ejercicios, así como resolución de dudas acerca de la lengua materna del usuario.

La aplicación se consagró como una de las 10 mejores Startup de 2008, además de recibir numerosos premios de publicidad y destacar gracias a sus gráficos y facilidad de acceso.

Bottom Navigation utiliza el máximo de pantallas disponibles(5):

Aprender, Revisar, Comunidad, Avisos y Perfil.

La barra de navegación desaparece al hacer scroll hacia abajo, y aparece al volver hacia arriba.

El botón de cada nivel se compone del icono, y el que está pulsado muestra la etiqueta.

Al pulsar sobre el botón de la pantalla activa, no vuelve al principio de la pantalla correspondiente.

Android Status Bar de colorPrimaryDark y **AppBar** de ColorPrimary

AppBar no contiene ningún icono. Sólo muestra el título de la Lección seleccionada y el progreso. Debajo, se sitúa otra barra permanente con el mensaje que invita a comprar la versión Premium.

Busuu

1. Palabra-a-palabra

Duolingo

Busuu

TinyCards

2. Cards para aprendizaje de vocabulario.

Se puede apreciar una serie de similitudes entre las diferentes aplicaciones que hemos visto hasta el momento.

4.1.4. Memrise

Memrise se define como unas de las aplicaciones más punteras del 2017 en cuanto a idiomas podemos hablar. Ha sido galardonada con el Premio Google a la Mejor Aplicación de 2017 durante el Mobile World Congress de Barcelona. Como particularidad, destacamos que los cursos son elaborados por los propios miembros de la comunidad de Memrise. El método que sigue presenta similitudes con otras aplicaciones, como Tincards o Busuu.

Al iniciar Memrise, se pide un usuario que puede ser creado a través de Facebook, opción que presentan otros juegos, después se debe escoger un nivel: principiante o avanzado.

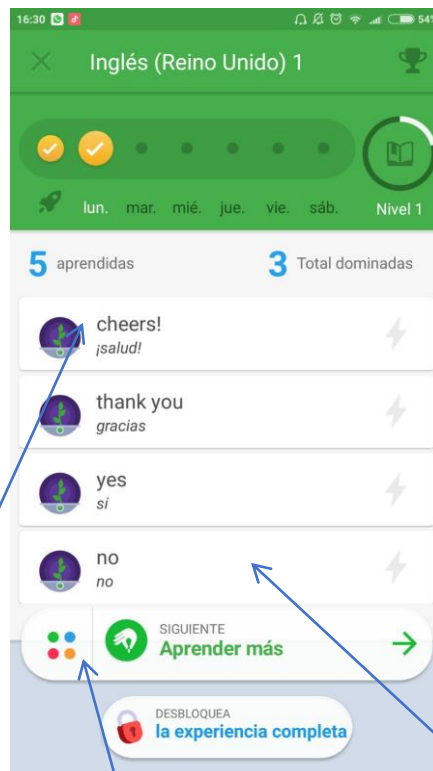
Memrise introduce el concepto de “mem”: se definen como imágenes que ayudan a definir y fijar los conceptos. A medida que se van completando fases del nivel, se suman puntos por aciertos. Por lo general, los ejercicios se separan en tres partes: escrito, oral y videos en nativo.

La aplicación esta disponible de manera gratuita, pero también tiene una opción de pago para mejorar la experiencia.

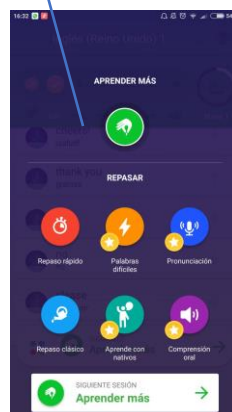
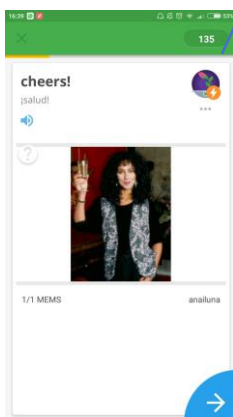
ActionBar con los indicadores de progreso diario, semanal y de nivel.

Scroll en la parte central que almacena las palabras aprendidas permitiendo volver a recordarlas.

Android Status Bar de colorPrimaryDark y **AppBar** de ColorPrimary



No es un **Floating Action Button** como tal, pero se encuentra en la posición de éste y contiene la acción principal de la pantalla: Aprender más sobre el idioma. La parte izquierda del botón nos lleva a una pantalla de diferentes opciones para repasar. La derecha nos dirige directamente a la siguiente fase del nivel.



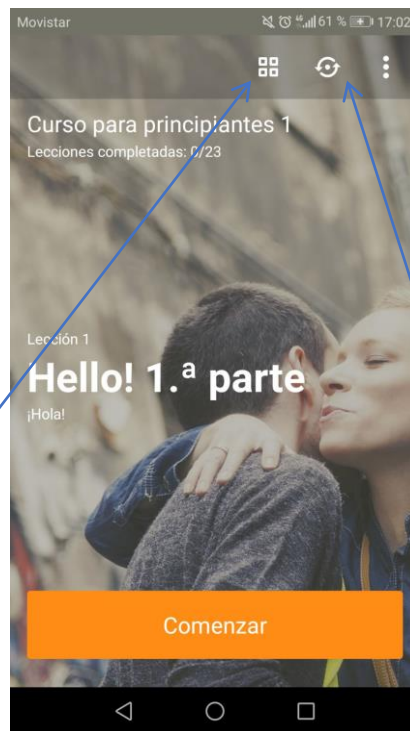
4.1.5. Babbel

Aparece en 2008 y se define como un software de aprendizaje de idiomas a través de plataformas online. Ofrece la posibilidad de aprender hasta un total de doce idiomas. A diferencia de las otras aplicaciones, ésta se descarga por módulos. De manera, que si deseamos aprender inglés solo descargaremos el módulo de inglés, ahorrando así espacio. Por el contrario, si tenemos varios módulos encontraremos una opción de acceso a cada uno de los diferentes idiomas. Al instalar la aplicación se realiza un pequeño cuestionario al usuario: edad(rangos), cómo has conocido Babbel, motivo de aprendizaje y el nivel en que deseas iniciar.

A diferencia de las otras aplicaciones que hemos visto anteriormente, la interfaz de ésta es ligeramente distinta. No tiene una página principal como tal, sino que el nivel actual es la página de inicio del idioma.

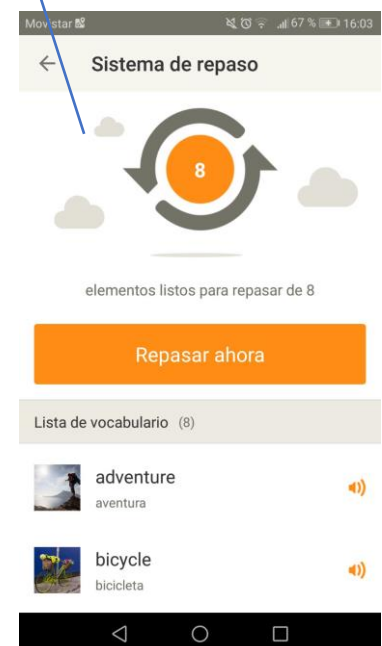
Los ejercicios que se realizan se basan en Listening de cuatro palabras, emparejar la traducción con la palabra, el writing de cada palabra o bien a través de teclado o escogiendo letras.

Botón Comenzar en grande, ocupando prácticamente el ancho de la pantalla. Acción de comenzar la lección actual.



Android Status Bar de color PrimaryDark y la **AppBar** de color transparente siguiendo las pautas de Material Design

En la **AppBar** se encuentran los botones principales: Cursos, Repaso y Menú de Desbordamiento.



4.2. Diseño de interfaz

4.2.1. Primeras ideas

Se propone a través de la Universidad el proyecto de Desarrollo de una Aplicación de Idiomas para el Ayuntamiento de la ciudad de Valladolid. Parte de este proyecto conlleva el rediseño de la interfaz de una aplicación existente, que se toma como base de ésta. En líneas generales la aplicación que se desarrolla está destinada a la práctica y mejora de la pronunciación de los lugares más emblemáticos de Valladolid.

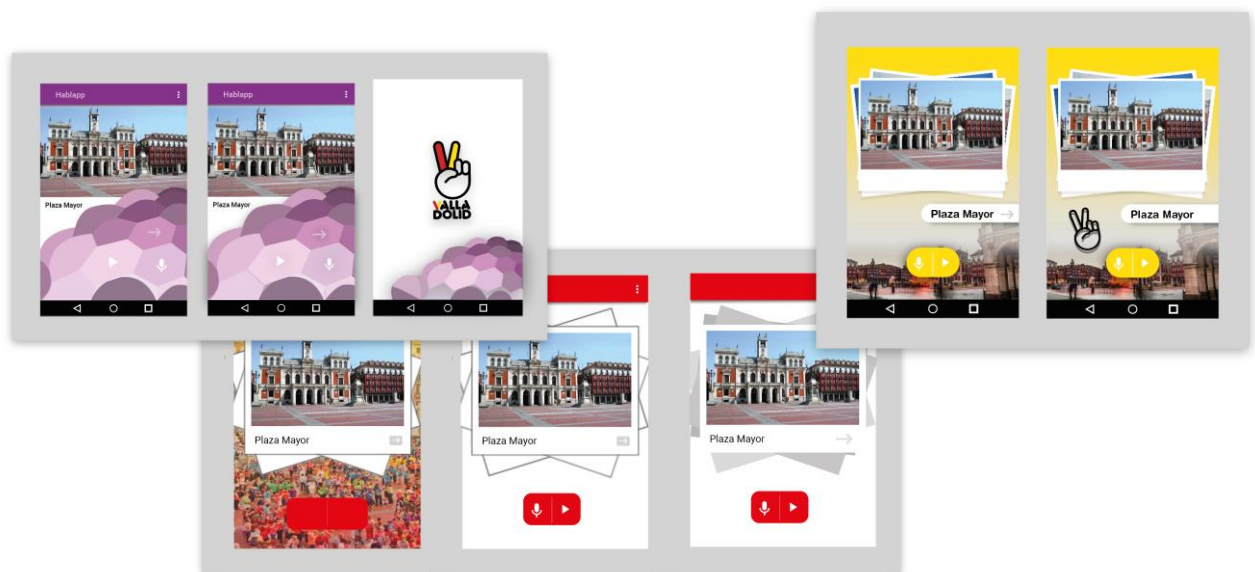
La línea gráfica de la aplicación va enfocada en la misma dirección que el resto del proyecto del Ayuntamiento. Tras obtener algunos folletos y manuales relacionados con el tema, se crean las primeras propuestas:

- 1) Basadas en los colores:
 - a) Siguiendo los colores del folleto de Spanish Valladolid: color rojo
 - b) Siguiendo los colores del folleto de Spanish Valladolid: color amarillo
 - c) Utilizando los colores emblemáticos de la ciudad: color violeta
- 2) Basadas en las formas:
 - a) Utilizando la geometría de la Cúpula del Milenio:

Hemos querido respetar algunos elementos que resultaron llamativos en la aplicación que se toma de base del proyecto, la cual tiene como uno de sus gráficos principales la Cúpula del Milenio. Siguiendo esa idea, se toma como geometría principal el hexágono.
 - b) Utilizando la línea gráfica de los folletos de Spanish Valladolid:



Los folletos informativos que ofrece el Ayuntamiento acerca del programa se caracterizan por incluir fotos de la ciudad utilizando el conocido formato físico de Polaroid. El fondo se basa en un degradado en amarillo que, junto con el rojo, son los colores predominantes en la línea gráfica. Por otro lado, el lema del programa es “Aprende español en Valladolid”. Así pues, los colores refuerzan la idea del lema a la vez que remarcan el carácter español de la ciudad.



4.2.2. Elementos de la interfaz

Tras un proceso de valoración, en el que participa equipo junto con la persona a cargo del proyecto por parte de la Sociedad Mixta para la Promoción del Turismo en Valladolid, se decide que la mejor opción es la interfaz amarilla basada en la idea del folleto.

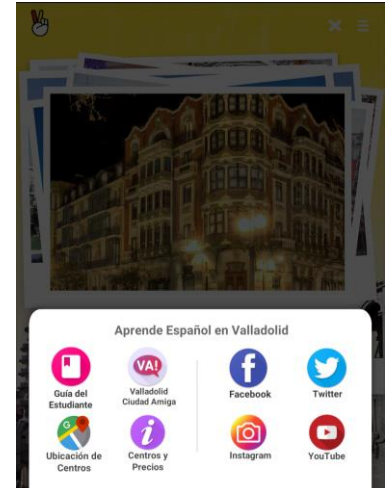
La aplicación se compone de tres pantallas: la pantalla de inicio, la pantalla principal del juego y el menú de enlaces.



31. Pantalla de Inicio



29. Pantalla Principal



30. Menú de desbordamiento

El icono de inicio de la pantalla principal ha sufrido una serie de cambios a lo largo del proceso de diseño. En un principio utilizamos la mano con el slogan en la zona inferior. Posteriormente, se consultó con el Ayuntamiento y se decidió que el logotipo de la aplicación debía integrar la V con el sombrero de la ñ. Se escoge el color rojo para el fondo del icono por analogía con el resto de la aplicación y tiene una leve sombra. El icono de inicio sigue la misma estética que los iconos del menú de enlaces.



32. Ideas iniciales



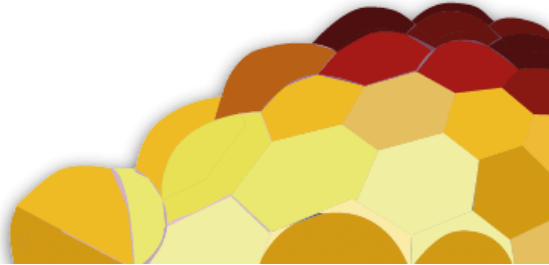
33. Resultado final del icono de inicio

4.2.3. Pantalla de Inicio

La pantalla de inicio en un principio fue concebida como una Splash Screen. Este tipo de elementos, que son bastante comunes al arrancar una aplicación, se componen de una imagen principal, como el logotipo de esta, un fondo sencillo y un mensaje claro y conciso. Se utilizan para cargar el contenido principal mientras que ofrecen una breve idea de la aplicación.

En nuestro caso, la pantalla de inicio se compone de:

- Fondo: El fondo de la pantalla de inicio se compone de varios elementos. El color escogido para el fondo es el blanco. Sobre éste, se coloca un fondo de rayos con un degradado radial amarillo-rojo. Se le aplica una textura orgánica para quitar la sensación de color plano. Mediante este fondo conseguimos enfatizar el logotipo que va situado en la confluencia de los rayos. Por otra parte, en el margen inferior derecho se sitúa la cúpula. Así pues, en la pantalla de inicio aparece una imagen de la cúpula vectorizada en sintonía con los colores principales de la aplicación.



35. Cúpula del milenio



34. Fondo de rayos



36. Logotipo

- Logotipo: Se toma como logotipo de la aplicación la mano en forma de V, ya que, junto con la V con el sombrero de la ñ, aparece de forma protagonista a lo largo de todo el programa. El logotipo está dotado de movimiento que simula pulso, que enfatiza e invita a tocar sobre él.
- Mensaje: en este caso, aquí es donde difiere un poco del concepto de Splash Screen como tal. Sobre un rectángulo de color amarillo con cierta transparencia y las esquinas redondeadas, se proyectan una serie de frases o eslóganes proporcionados por el Ayuntamiento. Con ellos se pretende animar a los usuarios a utilizar los servicios que presta el programa.

4.2.4. Pantalla Principal

La zona de juego de la aplicación se ubica en la pantalla principal. Por ser la más importante, su interfaz es algo más compleja, de manera que la separamos en dos partes: el fondo y los elementos interactivos. A mayores nos encontramos los elementos emergentes.

A. El fondo:

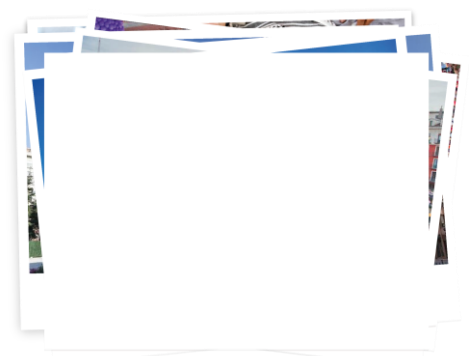
El fondo de la pantalla principal se compone de un degradado lineal de amarillo a un amarillo más tenue casi blanco. En la parte inferior de la pantalla se funde con una imagen de la Plaza Mayor de Valladolid. La imagen es proporcionada por el Ayuntamiento, para evitar así problemas de derechos de autoría. En la zona superior central es donde encontramos el marco donde se ubican las fotografías del juego. Simula un montón de fotografías en formato Polaroid, haciendo referencia a los folletos del programa.



39. Fondo degradado



38. Imagen inferior



37. Marco de fotos

B. Elementos interactivos:

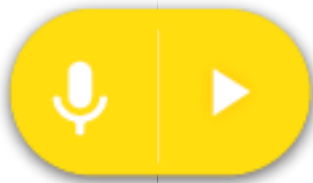
La aplicación se compone de numerosos elementos interactivos que permiten tanto jugar como avanzar y retroceder entre las diferentes pantallas.

- En la parte superior de la pantalla encontramos una barra con los distintos botones que permiten el desplazamiento entre pantallas. En la parte izquierda tenemos el botón que nos permite volver a la pantalla de inicio. Gráficamente es el mismo logotipo que en ésta. En la derecha nos encontramos dos botones: el primero nos lleva al menú de desbordamiento y el segundo es una X, que nos permite salir de la aplicación.



40. Barra Superior

- En la parte central nos encontramos los botones que nos permiten jugar:
 - El botón de siguiente para pasar a una nueva fotografía. La imagen actual que se muestra también tiene la función de pasar a la siguiente fotografía si se pulsa sobre ella.
 - El botón reproducir para escuchar la pronunciación de la palabra.
 - El botón grabar para pronunciar la palabra en cuestión.



43. Botón reposo

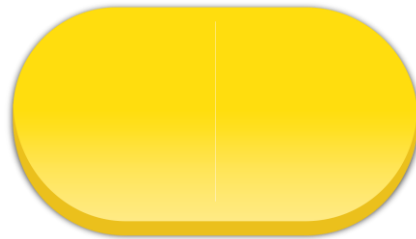


41. Botón pulsado



42. Botón siguiente

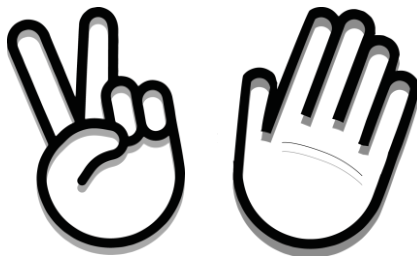
Siguiendo con lo estudiado sobre Material Design se decide que los botones de grabar y reproducir deben tener el suficiente protagonismo, ya que poseen las acciones principales de la aplicación. Se elige una estética similar al Floating Action Button, que se encuentra elevado por encima del contenido de la aplicación. Se decide poner un botón ovalado ya que, según las directrices de la guía de diseño de Android solo puede haber un FAB por pantalla, y en nuestro caso necesitaríamos dos. Así pues, se construye el botón siguiendo las pautas, pero modificándolas y adaptándolas a nuestro caso en particular. Para darle más protagonismo se decide remarcar la sombra arrojada.



44. Apariencia del botón de acciones principales

C. Elementos emergentes:

Existen dos elementos que no están visibles de manera permanente en la interfaz de la aplicación. Son animaciones que nos indican la valoración del ejercicio que hemos realizado. Si pronunciamos correctamente o de manera errónea la palabra que se muestra aparecerá una u otra. Se escoge como imagen para las animaciones una variación de la mano del logotipo inicial, indicando en cada caso un gesto acorde con el resultado. En un principio las manos eran de color negro, pero finalmente se opta por coger los colores de la aplicación: rojo para el gesto de error y amarillo para el gesto de bien.



46. Primera opción



45. Elección final

4.2.5. Menú de desbordamiento

El menú de desbordamiento se concibe como un menú desplegable que permanece oculto de forma natural. Incluye opciones que no son tan importantes como para

situarlas en una barra de acciones. En nuestro caso, el menú se puede dividir en dos zonas. La primera incluye la opción de compartir a las distintas redes sociales que posee el programa del Ayuntamiento, como Facebook o Twitter. La segunda opción incluye enlaces a las principales webs de interés si desea contratar alguno de los servicios de aprendizaje de español que ofrece la aplicación: Centros y Precios, la Guía del Estudiante ...

Los iconos de los botones se sitúan en un círculo del color principal de cada uno, junto con una leyenda en gris, situada en el centro y debajo de ellos. Los iconos se realizan con una herramienta que permite generar iconos, proporcionada de manera gratuita por Android y que se encuentra disponible en la web. Los iconos pueden proceder de imágenes prediseñadas de material, de imágenes personalizadas y de strings de texto. Con el Image Asset Studio se pueden crear iconos con las diferentes densidades generalizadas recomendadas para que el icono se vea apropiadamente en los diferentes tamaños de pantalla de los dispositivos.



47. Iconos realizados con Image Asset Studio



48. Menú desplegado

4.3. Componentes

A la hora de implementar todas las funciones que hemos descrito, tenemos que plantear los retos que supone la interacción entre el usuario y la aplicación, así como la respuesta que se espera de ésta.

Para realizar el análisis de una forma más concreta vamos a dividir el análisis en dos partes, siendo la primera la interacción del usuario con el interfaz de la aplicación. La segunda parte consta de la integración y uso de los diferentes elementos exteriores a la aplicación.

“El tamaño importa”. Tenemos que ser conscientes y precavidos a la hora de minimizar el tamaño de la aplicación, pues esto repercutirá en su funcionamiento. Como sabemos, se intenta minimizar al máximo el tamaño y el peso de los

dispositivos, esto significa que no se pueden permitir grandes discos de almacenamiento. Es un hecho que las memorias de los móviles se van llenando de datos, cuando el sistema está sobrecargado pide eliminar datos para liberar espacio. Este mensaje se traduce en realizar un chequeo rápido de las aplicaciones que no son de gran utilidad o que ocupan demasiado espacio. No nos interesa que la nuestra esté entre esas, de modo que el tamaño de la aplicación es algo que controlaremos a lo largo de todo el proceso de desarrollo. Esto plantea el primer reto que tenemos que superar a lo largo de toda la aplicación.

4.3.1. Retos de la pantalla de inicio

Como hemos visto anteriormente, la ventana de inicio se compone de diversos elementos. Podemos dividir los principales retos de la primera pantalla siguiendo el diseño de esta:

- El fondo: volvemos a hacer hincapié en el tamaño que tienen que tener los diferentes componentes. La cúpula, por ejemplo, es una imagen vectorizada que tenemos que tener en cuenta dado su tamaño. El color plano del fondo, según el entorno de desarrollo vamos a poder implementarlo de una manera u otra, siempre con el objetivo de minimizar el peso del proyecto.
- El logotipo de la mano: el logotipo de la mano tiene dos retos. El primero es el movimiento que le caracteriza. Tiene un movimiento de pulso que se mantiene constante mientras la pantalla esté activa. Este movimiento anima al usuario a pulsar sobre el botón para continuar. La función de avanzar a la pantalla principal cuando se pulsa sobre el mismo es el segundo reto que tiene el logotipo. Para que esta función esté habilitada debemos haber cargado antes todos los archivos exteriores necesarios para el funcionamiento de la pantalla principal.
- Pase de eslóganes: el pase de los distintos eslóganes que refuerzan la idea de la aplicación es el último reto que integra la primera pantalla. Cada eslogan debe estar visible un tiempo determinado. Una vez que se acaben, el pase debe de volver a empezar. Se requiere por parte del personal de Ayuntamiento que se incluyan las siguientes frases en la pantalla de inicio:
 - Aprende español en el corazón de España
 - Ven a Valladolid a mejorar tu conocimiento de español
 - Valladolid, tierra de escritores en el corazón de España
 - Valladolid, español cien por cien
 - Los mejores cursos de español... ¡en Valladolid!
 - Mejora tu nivel de español en Valladolid
 - Te esperamos en Valladolid para tu curso de español
 - El mejor español del mundo se habla en Valladolid
 - Valladolid, una ciudad moderna, segura y con mucha historia
 - Conoce Valladolid, ¡no esperes a que te lo cuenten!

4.3.2. Retos de la pantalla principal

La pantalla principal es la más compleja ya que está compuesta por multitud de elementos.

- El fondo está compuesto, como hemos visto anteriormente, por diferentes elementos que deben ser incluidos de forma adecuada para minimizar el tamaño de la aplicación.
- La ActionBar que se encuentra en la parte superior de la pantalla integra los pases hacia ambos lados de la pantalla, así como el cierre de la aplicación. El reto consiste en crear esos enlaces al resto de pantallas de la aplicación y que éstas se visualicen correctamente.
- Botón Reproducir: Botón encargado de la función de reproducir el audio. Para que no se colapse la aplicación será necesario que el comportamiento de los demás botones permanezca inactivo hasta que finalice el tiempo de reproducción. El botón debe cambiar su aspecto, tornándose oscuro, al ser pulsado.
- Botón Grabar: Es el encargado de activar la opción de grabar el audio. El micrófono del dispositivo recoge el audio del usuario, pronunciando en cada caso la palabra que se muestra en la pantalla. Al igual que en el botón de reproducir, los comportamientos deben permanecer inactivos hasta que se termine la acción. En caso de que la configuración indique que el reconocedor de voz está inhabilitado, el micrófono aparecerá desactivado. El tiempo de grabación debe de ser el adecuado para que grabe toda la palabra sin cortes ni interrupciones. El botón reproducir y el de grabar tienen el mismo estilo, por tanto, siendo su forma igual pero opuesta, deben de funcionar igual forma.
- Botón Siguiente: es el encargado de permitir el pase a la siguiente imagen. Debe permanecer inactivo mientras esté funcionando el reconocedor de voz o reproduciéndose el audio para evitar problemas de colapso. Cambia de color al ser pulsado, para reforzar la interacción entre el usuario y la interfaz. Por otra parte, también se contempla la opción de incluir el pase de imagen en la propia imagen, esto es, que la imagen mostrada en cada caso tenga función de botón. A menudo nos encontramos páginas web o galerías de imágenes que pulsando sobre la imagen actual pasa a la siguiente. Sería interesante incluir esta opción en nuestra aplicación, para un manejo más sencillo de la misma.
- Cuadro de texto: en el cuadro de texto se debe mostrar el título correspondiente de cada imagen. Las palabras se encuentran en un fichero de texto que está ubicado en un servidor. Debemos leer dicho documento para extraer las palabras. Debe cambiar cuando se pulse el botón de siguiente. Las palabras que aparezca en cada caso son las que el usuario debe pronunciar pulsando el botón de grabar.
- Animaciones: Las animaciones se lanzan al obtener un resultado de la transcripción del audio. Si la transcripción coincide con la palabra que reside en el cuadro de texto se lanza la animación de la mano amarilla, dando el visto bueno a la acción. Por el contrario, si al comparar la palabra que obtenemos con la que se muestra el resultado no coincide, se visualiza la animación de la mano roja en señal de error. La animación de ambas imágenes es la misma, un efecto latido como el del logotipo inicial. En este caso no se reproduce de manera indefinida, sino que se muestra durante unos segundos y desaparece para dar paso a la siguiente imagen.

4.3.3. Retos del menú de desbordamiento

El menú de desbordamiento, como hemos visto anteriormente, contiene los enlaces a las diferentes redes sociales y páginas de interés que posee el programa. Aparece de manera emergente desde el margen inferior de la aplicación. Esta animación debe ser muy breve. El planteamiento de su funcionamiento es el siguiente: en el contenedor se encuentran los botones y pulsando sobre ellos accedemos a los enlaces. El menú aparece superpuesto sobre la pantalla principal ocupando un tercio de ésta. El espacio sobrante está velado con un filtro negro con cierta transparencia, dejando ver la zona de juego. Este filtro tiene carácter de botón, pues al pulsar sobre él salimos del menú y volvemos a la pantalla principal. Si pulsamos sobre el botón físico del dispositivo también retrocedemos a dicha pantalla. En caso de pulsar sobre un botón, se redireccionará al buscador o en caso de tener la aplicación en cuestión, se abrirá dicha aplicación. En nuestra aplicación se cerrará automáticamente el menú de desbordamiento, volviendo a la pantalla principal. A continuación, se adjuntan las direcciones web de las páginas de interés que se deben incluir en cada icono:



"https://drive.google.com/open?id=1uLK4XVQBjuHHLkTBIK5lJurl_ePym_ID&usp=sharing"



"<https://www.instagram.com/ayuntamientovll/?hl=es>"



"<https://www.facebook.com/learnSpanishinVII>"



"<https://twitter.com/spanishinVLL>"



"<https://www.youtube.com/SpanishValladolid>"



"<http://www.info.valladolid.es/documents/20147/0/Guia%20Estudiantes%20Espa%C3%B1ol%20en%20Valladolid%202015/2e630aca-989d-1cd6-c530-70905f67ade1?version=1.0>"



“<http://www.info.valladolid.es/web/info/aprende-espanol>”



“<http://www.info.valladolid.es/documents/20147/0/Aprende%20Espanol%20en%20Valladolid/154fbd67-ece5-0534-4ac3-73ca7767f41d?version=1.0>”

4.3.4. Acceso a documentos de texto

Gran parte de la información de la aplicación se encuentra ubicada en servidores. Hay que acceder a ella de manera externa. Una parte muy importante se ubica en ficheros de texto. Esto hace que la aplicación sea independiente de los datos. Pudiendo así añadir o quitar información sin modificar el código de ésta. En este caso nos podemos poner en el supuesto en que el Ayuntamiento nos proporciona más fotografías de las que incluimos en un primer momento. Los datos de dichas fotografías, junto con el audio y el texto, se ubican en el servidor, permaneciendo el código de la aplicación invariable. La aplicación debe leer esa información.

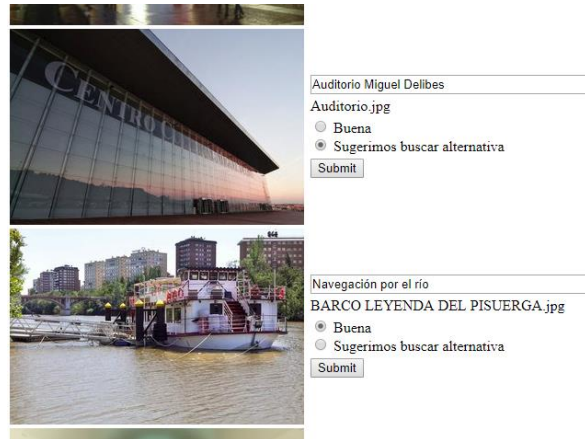
Concretamente hablamos de los ficheros de texto que contienen la información de las palabras, las posiciones del audio y la configuración de la aplicación.

- *palabras.txt* : Fichero de texto que contiene los nombres de las palabras. Siguen el mismo orden que el número de las fotografías. Las fotografías van desde imagen1.JPG a imagen54.JPG. De modo que la primera palabra se corresponde con la imagen1 y así sucesivamente. Esta relación se crea a través de una página web implementada específicamente para este uso, en la cual aparece una miniatura de la imagen junto con un cajetín de texto para introducir la palabra que deseamos. El fichero de texto se crea y se actualiza a través de ella. Se debe leer la palabra de tal manera que se corresponda con la foto que se está mostrando.
- *posicionesAudio.txt* : Fichero de texto que almacena la posición del audio de cada palabra. Las palabras han sido grabadas previamente por una actriz, en el orden en que aparecen en el documento de texto, esto es, que se encuentran colocados en el mismo orden que las palabras y las fotos. Lo mismo pasa con las posiciones del audio: encontramos dos datos por línea (separados por un tabulador) como por ejemplo 0.470000 1.800000. El primer dato es la posición de audio de comienzo de la palabra en cuestión, en este caso la primera. El segundo número es la posición final del audio. Estos datos son necesarios para el reproductor de sonido, que requiere el tiempo de comienzo y el de duración.
- *config.txt* : Fichero de texto que contiene la configuración de la aplicación. Son dos datos relacionados con el reconocedor de voz. El reconocedor de voz, como explicaremos más adelante, es un servicio de Google de pago. Como es lógico, necesitamos tener un interruptor que, en caso de necesidad, nos permita desactivar la opción del reconocedor. Contiene el dato del interruptor:

on/off y la clave encriptada de Google Speech API, que desencriptamos al leerla en la aplicación.

4.3.5. Acceso a fotografías

Como hemos mencionado con anterioridad, el Ayuntamiento nos proporciona una serie de fotografías para emplear en el juego. Por temas de derechos de autor, se prefiere esta opción a tomar nuestras propias fotos o cogerlas de internet. En un principio el total de fotografías disponibles para utilizar es de 64. Analizamos las fotografías y nos dimos cuenta de que algunas no cumplían unos requisitos mínimos para incluirlas en el juego, como baja calidad o desenfoque. Se crea la web que permite incluir el texto acorde con el contenido de cada foto y además permite señalar si la foto es correcta o sugerimos buscar una con mejores características.



49. Apariencia de la página web

Añadimos un título que nos parece que va en relación con la foto y valoramos si es buena o sería preferible otra. Después enviamos la dirección de la web a la persona a cargo del proyecto por parte del Ayuntamiento para tener una segunda opinión. En caso de tener que buscar una alternativa para la foto, que valore si nos puede proporcionar otro archivo relacionado con el tema. Tras este proceso de criba nos quedamos con 54 fotografías aptas para su uso.

Las fotografías se encuentran alojadas en una carpeta dentro de un servidor cuya dirección web es: <https://kere.eca-simm.uva.es/ayuntamiento/app/>

Cada fotografía tiene un número que la identifica y la posiciona en un orden. Este número es fijo y permanecerá invariable, ya que de él depende el audio y el texto. El nombre de cada una sigue la siguiente estructura:

<https://kere.eca-simm.uva.es/ayuntamiento/app/imagen> + número* + .JPG

*siendo número $\in [1, 54]$

Debemos cargar las imágenes desde el servidor, evitando así un gasto totalmente innecesario de los recursos y memoria del dispositivo. Las imágenes se almacenan en la memoria caché de este. En nuestro caso se plantea casi imposible cargar tal cantidad de fotografías como archivos internos de la aplicación, ya que esto elevaría muchísimo el tamaño de esta.

4.3.6. Acceso al audio

En un principio se utiliza el sintetizador de voz de Google para hacer pruebas, que lee la palabra en voz alta. Posteriormente se decide que la mejor opción es grabar las palabras con una voz real, en este caso, la de una mujer. Se realiza la grabación de las palabras en un mismo archivo, es decir, una detrás de otra con unos segundos

de separación. Con un programa de edición de audio se obtienen los segundos de inicio y fin de cada palabra. Estos momentos son los que se almacenan en el fichero de texto antes mencionado. Respecto al archivo de audio, se encuentra ubicado en la siguiente dirección:

<https://kere.eca-simm.uva.es/ayuntamiento/app/audio.mp3>

El reproductor de sonido debe cargar el audio sólo una vez. Debe reproducir el archivo comenzando en el segundo que corresponda con la palabra actual. Debe detenerse en el instante de fin de la palabra.

4.3.7. Uso de ASR – Automatic Speech Recognition Google Speech API

La API pertenece al grupo de Google Cloud, que pone a disposición del usuario la tecnología que emplean en sus productos. Permite a desarrolladores de aplicaciones utilizarla para convertir el audio en texto. Esta conversión se realiza gracias a las poderosas redes neuronales aplicadas en una API sencilla de utilizar. El audio puede proceder del micrófono de la aplicación, del control por voz o puede incluso transcribir archivos de texto. La transmisión de texto se realiza en tiempo real, según va avanzando la grabación. Otra facilidad que ofrece el servicio de Google es el reconocimiento de palabras clave, que permite personalizar las situaciones o los entornos.

Los archivos de audio pueden ser codificados como FLAC, AMR, PCMU y Linear-16. También existe la posibilidad de filtrar contenido inapropiado como palabras malsonantes.

- Precios:

“La API Speech de Cloud se tarifica por intervalos de 15 segundos de audio procesado después de los 60 primeros minutos, que son gratuitos. “

Este precio es válido para aplicaciones de sistemas personales, para dispositivos integrados, como, por ejemplo, coches o televisores, necesita consulta.

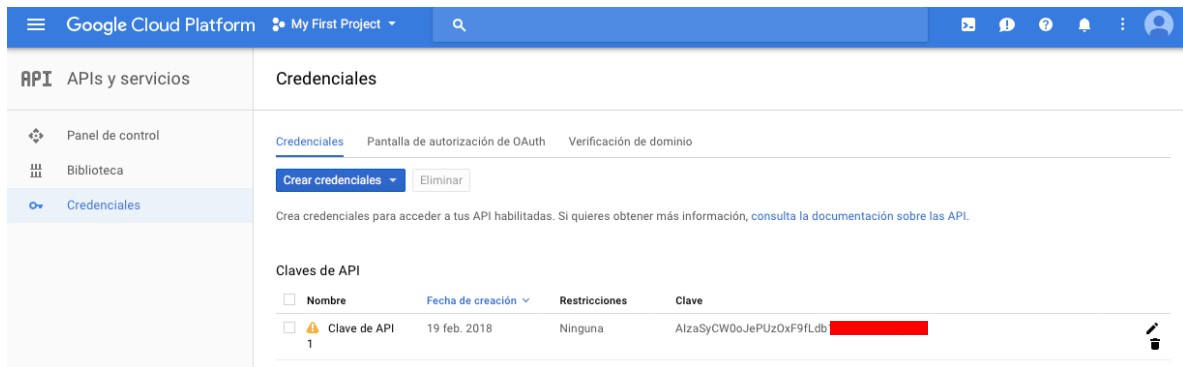
USO MENSUAL	PRECIO POR CADA 15s
Hasta 60 min	Gratuito
61 - 1.000.000 min	0,006 \$

- Usar Google Speech API:

Para utilizar Google Speech API debes tener una cuenta en Google Cloud Platform. Se necesita un correo Gmail. Ofrece una prueba gratuita, que a diferencia de otras aplicaciones que te ofrecen un tiempo, un mes por ejemplo, Google ofrece un crédito de 300\$, unos 200€ que te permite gastar en la API durante los próximos 12 meses. Tenemos que rellenar un formulario con nuestros datos: nombre y apellidos, dirección... A continuación, el tipo de pago (*):

*Google necesita los datos bancarios de una tarjeta de crédito/débito, donde asegura previamente que es sólo para comprobar que no eres un robot. No se realizará ningún cargo a esa tarjeta hasta que no se gaste el dinero que nos da en el tiempo que se indica.

Tras crear la cuenta, en la parte izquierda tenemos un menú desplegable: debemos ir a la opción de Productos y Servicios, y buscar la opción de Credenciales.



50. Captura de pantalla de las claves de API

En credenciales tenemos un apartado donde aparecen las Claves de API con la fecha de creación y la clave. Esta clave es la que utilizaremos posteriormente en el código de la aplicación para enviar el audio a Google y que utilice nuestra cuenta personal.

- Código del programa:

Para entender lo que realiza este servicio adjunto una breve explicación: nuestra aplicación envía a Google un audio que se recoge a través del micrófono del dispositivo, codificado en uno de los formatos que hemos señalado anteriormente e indicando el idioma. Google reconoce el audio y nos devuelve una serie de palabras que cree que pueden encajar con la que hemos mandado junto con el nivel de confianza de cada palabra.

```
- public static const
GOOGLE_URL:String="https://speech.googleapis.com/v1/speech:recognize?key=";

- public static const GOOGLE_API_KEY:String="AIzaSyCW0oJePUz0xF9FLdb*****";

- var url:String = GOOGLE_URL+_apiKey;
```

A través del código anterior se compone la URL de Google con la contraseña de nuestra sesión de Speech API. Tras recoger el audio y codificarlo, se crea una función con un objeto indicando las características de los datos que se van a enviar.

```
- var obj:Object ={ config:{encoding:"FLAC",sampleRateHertz:44100,languageCode:"es-ES",enableWordTimeOffsets:false,maxAlternatives:5},audio:{content:audioBase64} };
```

Después se crea una función que permite recoger los datos enviados por Google a través de una variable tipo JSONParser y los almacenamos en un array.

```
-var jsonParser: JSONParser = new JSONParser();

-var array_palabras: Array = jsonParser.parseGoogleSpeechJSON(ev.data); //data: recoge los datos que envía Google
```

En nuestro caso, comparamos las palabras con la nuestra. De este ejercicio se esperan dos respuestas, que recogemos a través de una variable de tipo boolean. Si la pronunciación de la palabra ha sido la correcta, recibiremos un "Bien" a través de la interfaz de la aplicación y, por tanto, quiere decir que Google nos ha devuelto una

palabra que coincidía con la que la aplicación requería. En caso contrario, la aplicación nos notifica que está mal lo que hemos pronunciado, es decir, que de las palabras que Google nos devuelve no coincide ninguna con lo que necesitábamos.

```
var listaPalabras:Array = new Array(5); //con 5 posibilidades de palabras
var textoReconocido:Object = JSON.parse(datos);
var resultados:Array = new Array();
    if (textoReconocido.results!=null)
        resultados = textoReconocido.results[0].alternatives as Array;
        for (var i:int = 0; i < resultados.length; i++){
            listaPalabras[i]=textoReconocido.results[0].alternatives[i].transcript;
        }
    return listaPalabras;
```

Esta función recorre el Array y compara las palabras que recoge en el array con la que cargamos nosotros y debe de coincidir. Almacena el resultado en textoCargado.

```
for (var i: int = 0; i < array_palabras.length; i++) {
    if (array_palabras[i] != undefined){
        if (array_palabras[i].toLowerCase() == textoCargado.text.toLowerCase()) {
            resultadoCorrecto = true; // Variable booleana que almacena si es correcto o no.
        }
    }
}
```

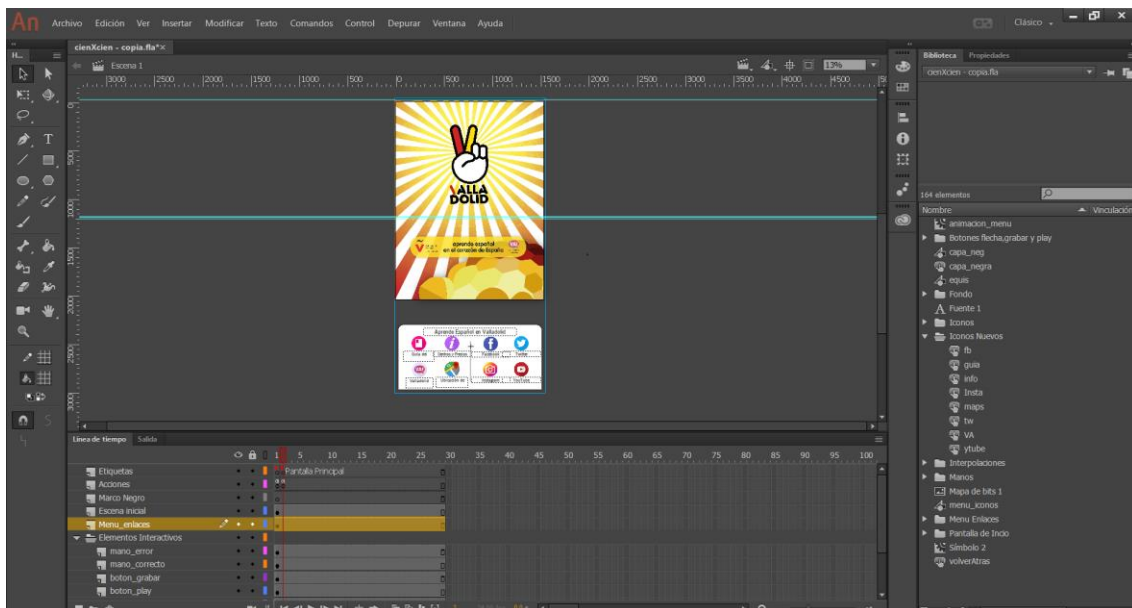

5. Implementación (comparativa ActionScript 3.0 vs Android Studio)

La aplicación de origen se desarrolla en Adobe Animate, utilizando como base una aplicación existente de la cuál reciclaremos algunos elementos. Se escoge Adobe Animate para el desarrollo del proyecto encargado por el Ayuntamiento, ya que el conocimiento y manejo de este es mucho mayor que de Android Studio. Posteriormente, se realiza el mismo proceso de desarrollo en Android Studio. El planteamiento de la interfaz de la aplicación, pese a tener un acabado final muy similar, cambia de manera radical de un entorno de desarrollo a otro. Debemos adaptar el contenido a cada entorno buscando una solución lo más similar posible.

5.1. Proceso de desarrollo en ActionScript 3.0

ActionScript 3.0 es el lenguaje de programación que se utiliza en los entornos de desarrollo proporcionados por Adobe Flash. Este tipo de programación pretende enfocar las aplicaciones de una forma más dinámica, permitiendo crear todo tipo de animaciones e interfaces que interactúen con el usuario. ActionScript 3.0 resultará más sencillo para aquellas personas que controlen lenguajes de programación orientada a objetos. Actualmente, Adobe Flash ha cambiado su nombre a Adobe Animate, pero su entorno de desarrollo es prácticamente igual.

Hablando en líneas generales, Animate es una aplicación destinada a la creación y manipulación de gráficos que suelen ser de naturaleza vectorial. Junto con los gráficos vectoriales encontramos elementos rasterizados. Además, suele ser común



51. Entorno de desarrollo Adobe Animate

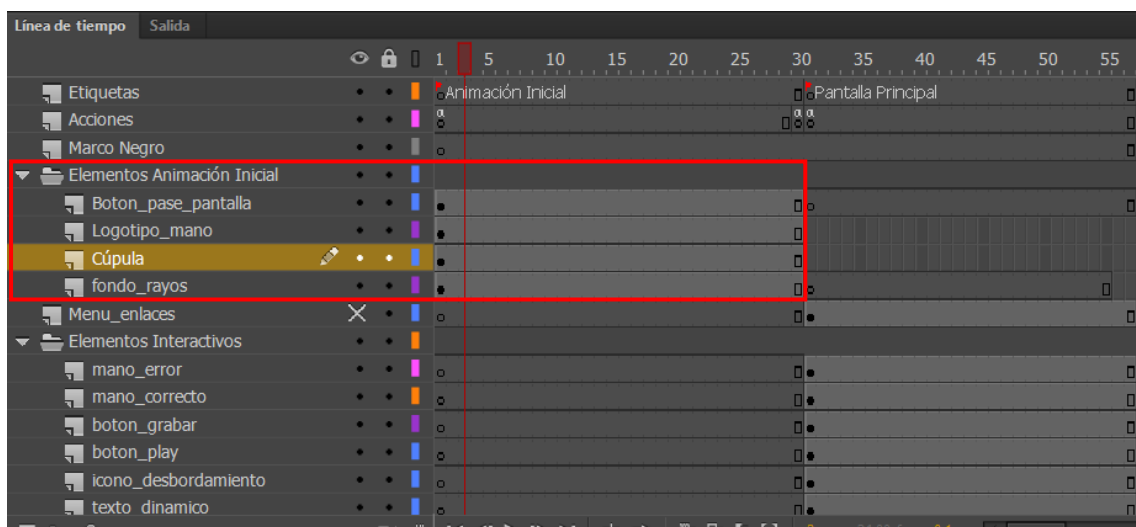
el uso de sonido, vídeo y líneas de código. En esta aplicación se trabaja sobre fotogramas, que representan instantes de tiempo. Estos fotogramas nos permiten dar movimiento a los objetos creando animaciones en dos dimensiones. Para facilitar la creación de animaciones, Animate cuenta con funciones como interpolaciones de movimiento o de forma. Una interpolación divide un movimiento en fotogramas para evitar tener que colocar en elementos en cada uno de ellos para dar la sensación de movilidad. Animate se compone de objetos, que pueden ser de diversas clases:

botones, movieclips, gráficos, vectores ... Cada objeto tiene su referencia en el archivo de gráficos swf

5.1.1. Pantalla de inicio

Se plantea la aplicación, como hemos descrito anteriormente, en tres pantallas diferentes. Vamos a tomar los diferentes retos que hemos planteado en el apartado anterior para explicar el desarrollo.

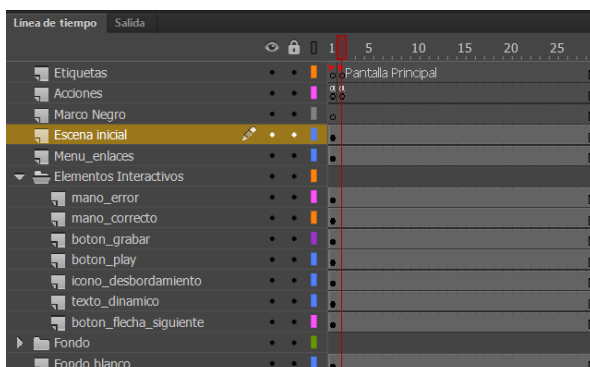
En cuanto a la pantalla de inicio, ha experimentado una serie de cambios a lo largo de todo el proceso de desarrollo. En un principio estaba situada en el primer fotograma de la animación. Pulsando sobre el botón de pasar a la pantalla de inicio, avanzábamos en la línea de tiempo. Como podemos ver en la línea de tiempo, la animación inicial ocupaba una serie de fotogramas identificados con una etiqueta. En la zona de las capas, encontramos una carpeta llamada “Elementos Animación Inicial” que contiene todos los componentes de la primera pantalla. Como podemos ver, estos elementos solo se muestran visibles en aquellos fotogramas que abarca la etiqueta de Animación Inicial.



52. Línea de tiempo principal

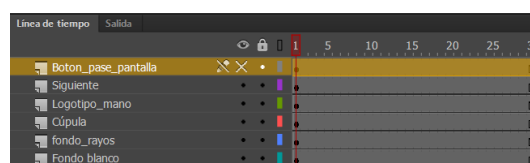
Posteriormente, se decide que necesitamos cargar los componentes de la pantalla principal mientras esté visible la pantalla de inicio e impedir el pase de pantalla hasta que todos y cada uno estén correctamente cargados para su uso. Se decide que es mejor convertir la pantalla de inicio en un movieclip y superponerla sobre la pantalla principal.

La escena de inicio se coloca sobre la pantalla principal, de tal modo que el pase de pantalla no es un pase de pantalla como tal, sino que oculta el visionado de la pantalla de inicio, es decir, del movieclip que la contiene. Dentro del movieclip la escena sigue siendo igual, los elementos son los mismos, pero en vez de estar colocados sobre la línea de tiempo



54. Línea de tiempo principal

que la contiene. Dentro del movieclip la escena sigue siendo igual, los elementos son los mismos, pero en vez de estar colocados sobre la línea de tiempo



53. Línea de tiempo del movieclip

principal se encuentran contenidos en el interior de este.

Elementos de la pantalla de inicio:

- El fondo: Se compone de los elementos mencionados anteriormente. Cada elemento se sitúa en una capa, como podemos apreciar en la línea de tiempo. Se colocan según vayan adelante o hacia atrás en el espacio. Por ejemplo, la capa que contiene el color blanco de fondo se sitúa en el último lugar, la cúpula, por el contrario, es el elemento del fondo que está situado por encima del resto.
- Botón de pase de pantalla: el botón de pase de pantalla también ha sufrido modificaciones a lo largo del proceso de desarrollo. En un principio se pensó que se debía hacer clic sobre el logotipo, el cual se encuentra parpadeando de manera constante. Luego llegamos a la conclusión de que es más práctico que toda la pantalla sea un botón, de manera que, toque donde toque el usuario, pasará a la escena siguiente. Se incluye una capa que integra un botón de iguales dimensiones que la pantalla. El logotipo deja de ser un botón y pasa a ser un movieclip que se encuentra situado por debajo del botón de pase de pantalla.
- Parpadeo del logotipo: el logotipo debe simular el latido de un corazón. Esto se crea mediante una interpolación clásica, la manera más antigua de Adobe Animate para crear animaciones. Posteriormente se crean las interpolaciones de forma y de movimiento. Como es una animación sencilla, un ligero cambio en el tamaño del logotipo, con la interpolación clásica es suficiente. Se crea, en unos pocos fotogramas, el cambio de tamaño de forma gradual y retorno a su tamaño original. Se debe incluir en las acciones del último fotograma la orden de que vuelva a reproducirse desde el fotograma 1. De esta manera se crea una animación en bucle.
- Carga del contenido de la pantalla principal: Como hemos mencionado, la función de esta pantalla es permitir cargar el contenido de la pantalla principal, de tal manera que cuando se pase a ella estén todos los elementos debidamente activos y cargados. Esta acción se realiza mediante el código situado en las acciones del primer fotograma, que activa el botón de pase de pantalla.

Se crean los cargadores de las URL con la dirección de cada archivo, vamos a coger como ejemplo el encargado de cargar las palabras que se muestran en el cajetín de texto. También se crean unas variables de tipo boolean que almacenan si el archivo se ha cargado correctamente o no. Están inicializadas en falso.

```
1. var cargadorPalabras: URLLoader = new URLLoader();
   var textosCargados:Boolean=false;

2. cargadorPalabras.load(newURLRequest("https://kere.eca-
   simm.uva.es/ayuntamiento/app/palabras.txt"));

3. cargadorPalabras.addEventListener(Event.COMPLETE, fCargado);
```

Cuando se haya completado el evento de cargar la URL del punto 2, se ejecuta la función `fCargado`. En cada caso, la función que se ejecuta es diferente (`limitesCargado`, `audioCargado`, ...). Dentro de cada función específica se encuentra la llamada a otra que hemos declarado previamente y es de tipo boolean: `todoCargado()`. Esta función almacena los valores de las variables que contienen el estado de carga de los archivos. Si todas las variables se encuentran en "true", la función devuelve "true" y con ello activa el Listener del botón de pase de pantalla. Si, por el contrario, a medida que se van cargando hay algunas en negativo y otras en positivo, devolverá "false". Se podrá pasar a la siguiente pantalla cuando en alguna de las llamadas a `todoCargado()`, estando todas las funciones individuales de carga en "true", la función devuelva "true".

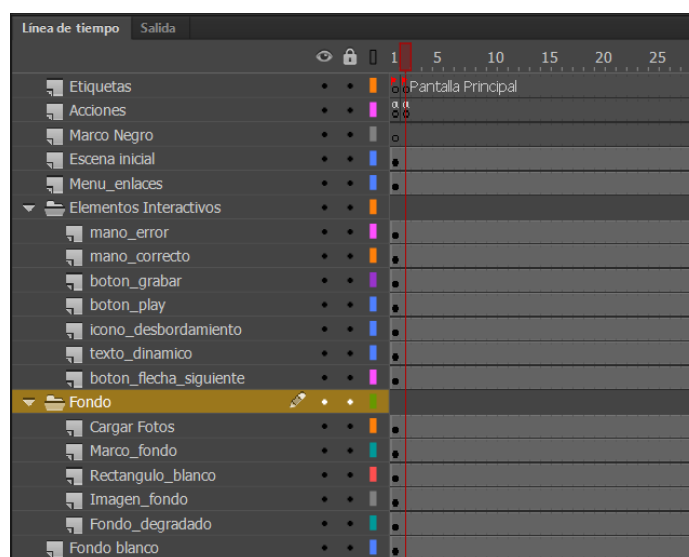
¿Qué hace cada función al cargar el archivo correspondiente?

- Audio: `onSoundLoaded()`. Cambia el estado de la variable de almacenamiento de estado. El audio ya está listo para ser utilizado.
- Configuración: `configCargado()`. El fichero de configuración contiene dos campos: El primero contiene si está activo o no el reconocedor de audio y el segundo contiene la contraseña de Google Speech API encriptada. En el fichero de origen se encuentran separados por una coma (,). Se realiza una lectura del archivo, almacena toda la información que se lea hasta que encuentre la coma en un Array llamado `campos []`. Almacena el primer dato en `campos [0]`, y la contraseña en `campos [1]`.
Se realiza una comprobación del estado del reconocedor, en caso de que esté apagado se cambia el estado del botón de grabar a inactivo. Por otra parte, se envía la contraseña desencriptada a la clase del reconocedor donde se requiere. Por último, cambia a "true" la variable de estado de carga del fichero de configuración.
- Límites de los audios: `limitesCargado()`. El fichero contiene los límites de cada palabra separados por saltos de línea (`\n`) y dentro de cada línea, el inicio y fin de cada audio separado por un tabulador (`\t`). Sabiendo esto, se realiza una primera lectura del fichero almacenando en un Array los datos separados por los saltos de línea. Después se recorre dicho Array separando los datos por el tabulador y almacenándolos en otro array donde `campos [0]` contiene el primer dato (inicio de palabra) y `campos [1]` el segundo dato (fin de palabra). Se asignan estos datos a `audioComienzo = campos [0]` y `audioDuracion = campos [1] - campos [0]`, ya que el reproductor de sonido necesita el inicio y la duración del audio. Se le da un valor de "true" al estado de carga de los límites.
- Texto: `fCargado()`. Las palabras se encuentran almacenadas en el fichero cada una en una línea. De manera que recorreremos el fichero leyendo los saltos de línea (`\n`) y almacenando cada palabra en un Array. La relación entre la posición de palabra y la de la fotografía será: $\text{posición} + 1 = \text{índice de la foto}$ que se debe mostrar, ya que las posiciones del Array empiezan en cero y los índices de la foto empiezan en el número 1. Se genera una secuencia de números aleatorios para lanzar la primera fotografía junto con el texto correspondiente. Se coge el primer número y se le aplica la lógica de los índices para que se corresponda la foto y el texto. Se carga la imagen. Posteriormente se carga el texto. Finalmente se da el valor "true" a la carga de las palabras.

- Función de números aleatorios: Esta función crea una secuencia de números aleatorios cada vez que se inicia la aplicación. Dicha secuencia será la que se utilice para establecer el orden de visionado de las fotografías. En la función de los números aleatorios se introducen dos parámetros de entrada, que son el mínimo y el máximo de la secuencia, en este caso son 1 y 54. De esta manera se crea una secuencia que tendrá 54 números siendo el más pequeño el 1 y el mayor el 54.
Junto con la función de los números aleatorios se crea otra función llamada “usados”. Esta función tiene la misión de comprobar si el número aleatorio que se crea ya ha salido antes, ya que los números que se van usando se almacenan en un array. Se crean números aleatorios hasta dar con uno que no esté en la secuencia de usados, en cuyo caso se transfiere ese dato donde sea necesario y se almacena dicho número en la secuencia de usados.
- Pase de eslóganes: En el centro de la pantalla se ubica un rectángulo de bordes redondeados y color amarillo anaranjado en el que se muestran los distintos eslóganes. En este caso, el pase de eslogan se realiza mediante un movieclip. Dejando como base el rectángulo amarillo a lo largo de todos los fotogramas, se va cambiando el texto que reside en el cuadro cada aproximadamente dos segundos. Cada texto se encuentra ubicado en una capa. El final de un eslogan se solapa con el comienzo del siguiente, de manera que no hay ningún instante de tiempo en el que la caja esté vacía. En el último fotograma reside una línea de código, en la zona de las acciones, que indica que cuando acabe de reproducirse el movieclip debe volver a empezar en el primer fotograma. De esta manera conseguimos que el pase de eslóganes sea dinámico todo el tiempo que se esté visionando.

5.1.2. Pantalla Principal

Como consecuencia del cambio de planteamiento de la pantalla de inicio, la pantalla principal también sufre modificaciones. Como hemos mencionado en el punto anterior, la pantalla principal se mantiene en el escenario en todo momento, ya que la pantalla de inicio aparece y desaparece posicionándose sobre ella.



55. Línea de tiempo principal

Los elementos aparecen ordenados en carpetas del mismo modo que los hemos explicado anteriormente. En una carpeta encontramos los elementos que integran en fondo y que no tienen carácter interactivo. Otra carpeta contiene los elementos de la aplicación que permiten al usuario jugar e interactuar con la misma.

Los elementos que componen el fondo se colocan en la escena siguiendo los diseños iniciales.

- LetterBox: Como sabemos, existen en el mercado dispositivos con diferentes tamaños de pantalla. Por las características de la aplicación, se decide que la pantalla se escale a partir de unas proporciones prefijadas 1536 x 2048 píxeles. La aplicación tiene una relación de aspecto 4:3, de manera que todas aquellas pantallas que sigan otras relaciones de aspecto modificarán el visionado de la aplicación. Se incluirán franjas negras en las zonas que la aplicación no cubra. Esta técnica de ajuste se conoce como letterbox. Son de color negro dado que el color del escenario de fondo es el negro. A la derecha, podemos observar el resultado en una captura de pantalla de un dispositivo con una relación de aspecto 16:9.



56. Franjas negras (LetterBox)

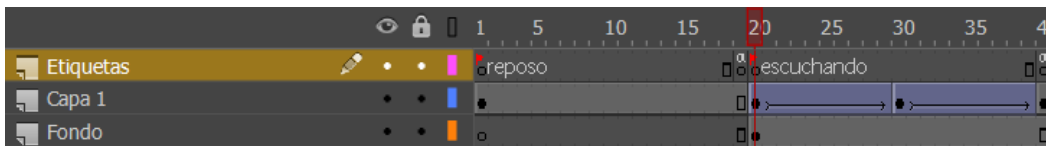
Los elementos interactivos de la pantalla principal se han resuelto de la siguiente manera:

- ActionBar: La barra de botones que se encuentra en la parte superior de la pantalla nos permite viajar a través de las pantallas que integran la aplicación, así como cerrarla.
 - o Volver al Inicio: Se crea un botón con la imagen del logotipo. En las Acciones del primer fotograma se implementa una función llamada "voy_inicio" que se ejecuta cuando se pulsa sobre el botón que hemos mencionado anteriormente. Esta función cambia la visibilidad de la escena de inicio, es decir pasa de `vistaInicio.visible=false` a `vistaInicio.visible=true`. También ordena al movieclip del pase de esloganes que empiece a reproducirse en el fotograma 1.
 - o Cerrar la aplicación: El botón que nos permite salir de la aplicación es un aspa de color blanco. Al pulsar sobre él, se ejecuta una función llamada `cerrarAPP`. `CerrarAPP` contiene la orden `NativeApplication.nativeApplication.exit()`, una llamada al método `exit()` que cierra la aplicación junto con todos los procesos que se estén ejecutando en ese momento.
 - o Ir al menú de desbordamiento: Unas líneas horizontales paralelas de color blanco, que normalmente son conocidas por indicar que pulsando sobre ellas se despliega un menú, indican la existencia de este. La función "abrirMenu" cambia la visibilidad del menú de desbordamiento, que se encuentra situado encima de la pantalla principal. También indica que debe reproducirse el movieclip del menú desde el fotograma 1. El planteamiento del funcionamiento de este botón es similar al de la pantalla de inicio.

- El botón reproducir: botón que permite reproducir el audio. Como hemos visto anteriormente, el audio se carga en el fotograma 1 de la línea de tiempo principal. De manera, que cuando estamos en la pantalla principal el audio ya está listo para ser usado. Para reproducir el sonido, además del audio, necesitamos los datos de los límites de tiempo cada palabra. Cuando se pulsa sobre el botón se ejecuta una función llamada `onEscucharMP3()`. En esta función se crea un temporizador de naturaleza `Timer`. A este objeto se le indica el tiempo de duración de la palabra, multiplicado por 1000 ya que el tiempo de duración está en segundos y el objeto necesita milisegundos, y además se le suma un margen de 200 milisegundos. Se inicia el `Timer` y se comienza a reproducir el audio con un objeto `channelMP3`. Se indica el tiempo de inicio, multiplicando otra vez el dato por 1000, para pasarlo a milisegundos. Por último, se cambia la apariencia del botón viajando al fotograma de “escuchando” y se desactivan los comportamientos de los botones para que no se puedan realizar dos acciones a la vez, como grabar y escuchar.

Hacemos una llamada al objeto `timer TimerEvent.TIMER_COMPLETE`, `stopSound`. Esto quiere decir que cuando se cumpla el evento, cuando pase el tiempo estipulado como duración que hemos declarado previamente, se ejecutará la función que le sigue: `stopSound`. La función para el reproductor de sonido y devuelve los comportamientos a los botones.

El diseño e implementación del botón se realiza a través de un `movieclip`. A pesar de ser un botón, hemos querido que durante el tiempo de reproducción se anime el símbolo de play y cambie ligeramente de tamaño para crear “feedback” entre la aplicación y el usuario.



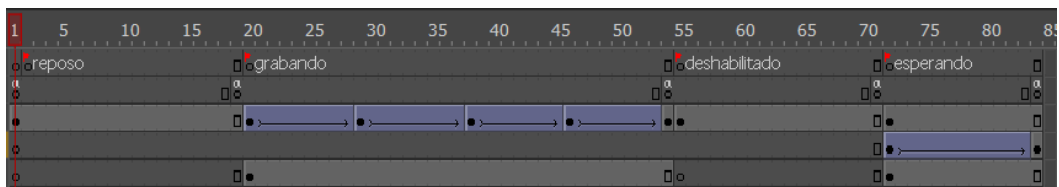
57. Diferentes estados del `movieclip`

- El botón grabar: una vez que se ha escuchado la pronunciación del audio es el turno del usuario. Pulsando sobre el botón grabar se activa la función `onGrabar()`. Primeramente, se cambia la apariencia del botón a “grabando”, realizando el mismo cambio de tamaño que el botón de reproducir, y se desactivan los comportamientos. Se inicia la grabadora con el método `recorder.startRecord()`. Se crea un listener que lanzará una función que se iniciará al terminar el evento, esta función se llama `onFinGrabacion` y para la grabadora cuando pasa un tiempo definido como constante de dos segundos y medio. Después el micrófono cambia su estado a “esperando” donde podemos visualizar una barra subiendo y bajando que indica que estamos esperando a la respuesta del reconocedor de voz. La aplicación manda el resultado codificado recogido de la grabación al reconocedor de voz de Google. Posteriormente se recogen las cinco transcripciones que Google Speech cree que se han podido pronunciar y las almacenamos en un array. Entonces se procede a comprobar si algún resultado de los obtenidos por Google concuerda con la palabra que identifica la fotografía. La comprobación se realiza recorriendo el array que almacena las palabras comparándolas una por una con la palabra en concreto, en caso de que coincida una variable de

tipo boolean recoge el resultado cambiando su estado a “true”. Finalmente, mediante un bucle se lanza la animación de la mano correcta o la de error, dependiendo del estado de la variable que recoge el resultado. El botón vuelve a cambiar su apariencia a “reposo” y se activan los comportamientos para continuar.

El diseño de este botón sigue la línea del anterior. Se añaden dos estados más, ya que el proceso del reconocedor lo requiere. Se pretende que el usuario en todo momento sea consciente de que la aplicación está funcionando o esperando una respuesta. Como hemos mencionado existen los estados de “reposo” y “grabando”, y se añaden otros dos: “deshabilitado” y “esperando”.

El estado “deshabilitado” se activará cuando al leer el fichero de la configuración de la aplicación el reconocedor este “off”. El estado “esperando” se reproduce mientras el reconocedor espera la respuesta del servicio de Google.



58. Diferentes estados del botón grabar

- El botón siguiente: La flecha que aparece a la derecha del texto permite al usuario avanzar a la siguiente fotografía sin necesidad de pronunciar bien la palabra actual. A lo largo del desarrollo de la aplicación se añade como zona activa de este botón aquella donde se muestra la fotografía. Así pues, si pulsamos encima de la fotografía actual pasaremos a la siguiente. La función que realiza este comportamiento se llama “inicializar”. Utilizamos para cargar el contenido fotográfico un Loader. Se coge un número aleatorio de la secuencia generada anteriormente. Aquí se utiliza una variable auxiliar inicializada a cero que va incrementando cada vez que se utiliza un número nuevo. Se compone la URL a cargar de la siguiente manera: "https://kere.ecasimm.uva.es/ayuntamiento/app/imagen"+ fotoMostrada³ + ".JPG" Se invoca un listener para que, cuando se complete la acción de cargar la fotografía, se lance una función que muestra el texto correspondiente en el cajetín e inicia el reproductor con el sonido de esa palabra. Recordamos que cada vez que se pasa a una palabra nueva se escucha la pronunciación de la palabra sin necesidad de pulsar el botón de reproducir.
- El texto: como ya hemos dicho anteriormente, el texto se lee del fichero que se encuentra en el servidor y se almacena en un array. La posición del texto se corresponde con la de las fotografías, pero al estar almacenado en un array se corresponden de la siguiente manera: imagen(i).JPG -> textoImagen[i-1], ya que la primera posición de un elemento en el array es la cero. El texto del cuadro donde se muestran los títulos de las fotografías es dinámico y tiene un tamaño de 65 p.

³ fotoMostrada: es el nombre de la variable que almacena el número generado aleatoriamente. Se deja este nombre en el ejemplo por analogía con el código de la aplicación.

- Animaciones de las manos: Las animaciones de las manos se lanzan al obtener un resultado de la pronunciación del usuario. Como ya hemos mencionado, existe una variable que recoge ese resultado al ir cotejando las diferentes posibilidades con la palabra de origen. En un principio está inicializada en negativo. Ambas animaciones son similares, realizan un recorrido relativamente corto en el tiempo, lo justo para que el usuario sea consciente del resultado de su ejercicio. Duran 1,7 s. Se muestran en pantalla a la vez que se oculta la fotografía actual. Son de naturaleza movieclip y realizan un movimiento bastante similar al descrito anteriormente, parpadean dos veces, aumentando de manera un poco más notable su tamaño, y se ocultan. Son dos animaciones diferentes, con nombres y ubicaciones diferentes: `mano_error- error` y `mano_bien- correcto`, pero en todo momento nos referimos a ellas de manera conjunta porque realizan la misma acción, salvo que su mensaje es opuesto. De manera natural aparecen ocultas en la escena principal. Cuando se lanza la orden de reproducir el movieclip correspondiente lo hace desde el fotograma 1, y una vez que llega al fin encontramos un `stop()`, no se reproduce de manera indefinida como en otros casos.

5.1.3. Menú de desbordamiento

El menú de desbordamiento de la aplicación contiene los enlaces a las diferentes redes sociales y páginas de internet. Estos enlaces están contenidos en botones, a través de los cuales accedemos a los enlaces.

La idea del menú es que emerja colocándose de manera superpuesta a la pantalla principal. Sobre la pantalla principal creamos una capa de color negro con un alfa, que nos permita ver el fondo, pero aplicando una transparencia negra para darle más importancia al menú, dejando así la pantalla principal en un segundo plano.

El diseño del menú consiste en un cuadro de color blanco, con una suave sombra arrojada. En el interior el menú se divide en dos zonas: en la parte derecha colocamos los botones que nos llevan a las páginas web relacionadas con el programa de la aplicación. En la zona izquierda situamos aquellas redes sociales de las que dispone el programa, junto con el enlace del canal de YouTube Aprende español en Valladolid.

Accedemos al menú desde la pantalla principal, pulsando sobre el botón que hemos explicado anteriormente. El menú se encuentra en una capa superpuesta a la pantalla principal. Haciendo clic sobre el botón que abre el menú lo que realmente hacemos es cambiar el estado de la visibilidad a “true”, así como empezar a reproducir el movieclip desde el primer fotograma. El menú está contenido en un movieclip que recoge el desplazamiento de este. Se despliega progresivamente desde la parte inferior hasta ocupar más o menos el tercio inferior de la pantalla. Dentro del movieclip está el gráfico que contiene el rectángulo blanco y los botones. Este gráfico es el que se desplaza en el movieclip, hasta quedar visible en la pantalla. El movieclip se para quedando este totalmente visible. Para salir del mismo, podemos utilizar el botón físico del terminal (back o volver). Otra opción que implementamos para salir del menú es el botón de la capa negra. Por analogía con otras aplicaciones, al tocar el fondo que se encuentra en segundo plano, volvemos a la pantalla que se encuentra en ese plano. Esta idea la trasladamos a la aplicación de manera que, haciendo clic en el fondo negro, volvemos a la pantalla principal.

Esto se implementa dotando a la capa de naturaleza botón, de esta manera cuando hagamos clic en cualquier parte de ella se ejecute una función que se llama cerrarMenu que cambia la visibilidad del menú a falso.

Cada botón integra el enlace a una página web. Todos siguen la misma estructura de ruta, cambiando el nombre de la función que contiene la dirección web. También se activan los comportamientos de los botones de la pantalla principal, que han sido desactivados al pulsar el botón que despliega el menú. Los iconos son botones que tiene dos estados. Cuando están en reposo muestran sus colores sin ningún tipo de filtro. Cuando nos situamos sobre ellos, así como cuando pulsamos, un alfa negro se sitúa sobre la zona activa para proporcionar feedback al usuario. La estructura del código de cada botón es la siguiente:

```
animacion_menu.menu_iconos.nombre_boton.addEventListener(MouseEvent.CLICK,
pagina_ClickToGoToWebPage);

function nombre_boton_ClickToGoToWebPage(event: MouseEvent): void {
    navigateToURL(new URLRequest("http://paginaweb.com "), "_blank");
    animacion_menu.visible = false;
    activaComportamientos();
}
```

5.2. Reutilización de elementos en Android Studio

Se realiza la migración de la aplicación Aprende español en Valladolid al entorno de desarrollo Android Studio. Un aspecto que se tiene en cuenta a la hora de realizar este proceso es la reutilización y la eficiencia a la hora de trabajar. En este sentido, hay un gran número de elementos que componen tanto la interfaz como el código que son reutilizables. Aunque cada entorno trabaje con un lenguaje de programación diferente⁴, la lógica de la programación y los elementos de la interfaz son válidos tanto en una plataforma como en otra, realizando pequeñas modificaciones.

1) Elementos de la pantalla de inicio:

Como hemos mencionado anteriormente trabajar de manera inteligente implica saber que elementos hemos de reutilizar de un proyecto a otro. En la pantalla de inicio hemos reutilizado varios elementos, entre los que se encuentran el fondo de rayos o el logotipo de la mano junto con el eslogan. También hemos reutilizado la imagen de la cúpula vectorizada. El fondo de color blanco se declara, como en Animate, como color de fondo del escenario de la aplicación. A continuación, podemos ver las capturas de la pantalla de inicio de ambas aplicaciones, donde apreciamos que los elementos mencionados son los mismos en ambas.

⁴ Lenguaje de programación diferente: Android Studio utiliza Java mientras que Animate programa en ActionScript 3.0



60. Aplicación desarrollada en Adobe Animate



59. Aplicación desarrollada en Android Studio

2) Elementos de la pantalla principal:

La pantalla principal integra un gran número de elementos gráficos, de los que, en comparación con otras pantallas, se han reutilizado bastantes menos. Podemos destacar como elementos comunes del fondo a ambas aplicaciones la imagen de la Plaza Mayor de Valladolid y el marco de fotos que simula un montón de fotos Polaroid.

Son dos elementos importantes ya que caracterizan la pantalla principal y no se considera la opción de sustituirlos o eliminarlos. Se aprecia una expansión de los elementos, ya que Android Studio permite adaptar los elementos a la relación de aspecto de la pantalla. Podemos notar claramente los elementos distribuidos de manera diferente, pues se gana el espacio que en Animate ocupan las franjas de color negro.

Respecto a los elementos interactivos:

- La colocación de los elementos de la ActionBar situada en la parte superior de la pantalla permanecen invariables, al igual que sus acciones. No así los iconos que la integran.
- Los botones que activar el reproductor y la grabadora de voz, se ha mantenido la misma estética, pero han sufrido modificaciones de acuerdo con las especificaciones del entorno.

Así pues, la interfaz tiene una apariencia similar a la otra, pero con modificaciones notables. Se puede decir que la colocación de los elementos guarda relación con la aplicación de origen, no así su apariencia.

5.3. Similitudes

Para realizar el trabajo de la manera más eficiente posible se han reutilizado elementos, tanto a nivel gráfico como a nivel de código. No hemos creado una reproducción exacta de la aplicación en dos entornos distintos, si no que se aprecia la esencia de cada uno.

5.3.1. Similitudes en la interfaz

Como ya hemos mencionado, una parte importante de este proceso de migración es la adecuación de objetos de un entorno de desarrollo a otro. Aunque la implementación de cada aplicación sea distinta, existen numerosos elementos en común. El diseño debe respetarse en ambas aplicaciones. Los botones guardan una similitud gráfica, a simple vista parecen iguales pese a estar contruidos de manera distinta, de acuerdo con la eficiencia del entorno.

Icono de Inicio: Como hemos visto anteriormente, la imagen del icono de inicio es la V con el sombrero de la "ñ" sobre un fondo rojo. Este icono se genera a través de la aplicación Android Asset Studio. Gracias a este servicio, obtenemos un conjunto de iconos con la resolución adecuada para cada densidad de pantalla (mdpi, hdpi, xhdpi, xxhdpi y xxxhdpi). Este icono lo reutilizaremos de una aplicación a otra.



61. Icono en diferentes resoluciones

- Animate: En la configuración de la publicación donde generamos el archivo .apk, tenemos un apartado que dice icono. Dentro de ese apartado podemos introducir diferentes tamaños de icono, con mostrar uno es suficiente. Se mostrarán de un tamaño u otro según la densidad de la aplicación.
- Android Studio: Debemos añadir la carpeta que contiene los iconos entre los recursos de la aplicación. En nuestro caso, la colocamos en res/mipmap – density. Dentro de cada carpeta se ubica un archivo con la resolución correspondiente y de igual nombre. A la hora de utilizar el icono lo debemos declarar en el archivo Manifest de la aplicación.

```
<application
```

```
    android:icon="@mipmap/iconoinicio"  
    android:label="@string/app_name"
```

5.3.2. Similitudes en el código

Una parte muy importante a la hora de realizar de nuevo la aplicación ha sido la reutilización del código. Debemos distinguir entre el lenguaje de programación que utilizamos, que ya hemos dicho anteriormente que es distinto, y la lógica que se utiliza para construirlo. Así pues, se reutilizan una serie de funciones de las que vamos a hablar a continuación:

- 1) Función de numero aleatorios y número repetidos: Es la función que se encarga de generar una secuencia de números aleatorios que marcará el orden en que se vayan lanzando las fotografías. Se declaran las variables de manera acorde a cada lenguaje de programación, pero la lógica de las funciones es la misma:


```

public int Aleatorio(int max, int min) {
    if (usados.size() != (max - min + 1)) {
        int num;
        boolean repe;
        do {
            num = (int) Math.floor(Math.random()
* (max - min + 1)) + min;
            repe = Repetido(num);
        } while (repe != false);
        usados.add(num);
        return num;
    } else {
        return 0;
    }
}

```

```

function aleatorio(min: Number, max: Number): Number {
    if (usados.length != (max - min + 1)) {
        var num: Number;
        var repe: Boolean = false;
        do {
            num = Math.floor(Math.random() * (max - min + 1))
+ min;
            repe = repetido(num);
        } while (repe != false);
        usados.push(num);
        return num;
    } else {
        return 0;
    }
}

```

62. Función Aleatorio en AS3 y en Java

```

public boolean Repetido(int num) {
    boolean repe = false;
    for (int i = 0; i < usados.size(); i++) {
        if (num == (usados.get(i))) {
            repe = true;
        }
    }
    return repe;
}

```

```

function repetido(num: Number): Boolean {
    var repe: Boolean = false;
    for (var i = 0; i < usados.length; i++) {
        if (num == usados[i]) {
            repe = true;
        }
    }
    return repe;
}

```

63. Función Repetido en AS3 y en Java

- 2) Extracción de los datos del fichero de texto y almacenamiento en la aplicación: Realizamos esta acción para extraer las palabras de texto y los límites del audio. Recorremos los ficheros de texto y almacenamos los datos en vectores en ambos casos.

```

final String[] palabrasFichero =
parseHTML(params[0]).toString().split("\\r?\\n");
for (String palabra : palabrasFichero) {
    palabras.add(palabra);
}
for (int i=0;i<=limitesPalabras.size()-1;i++){
    String linea = limitesPalabras.get(i);
    String[] campos= linea. split("\\t");

    String audioPrincipio= campos[0];
    String audioFin=campos[1];
}

```

```

limitesCargados=true;
audioLimites = String(e.target.data).split('\n');
var linesNum: int = audioLimites.length-1;
for (var i: int = 1; i <= linesNum; i++) {
    campos=audioLimites[i-1].split('\t');
    audioComienzo[i-1]=campos[0];
    audioDuracion[i-1]=campos[1]-campos[0];
}

```

64. Función que recoge los límites de los audios de las palabras en AS y en Java

- 3) Funciones que activan/desactivan los comportamientos de los botones: en cada entorno se desactivan los comportamientos de una manera diferentes. Para activarlos/desactivarlos hacemos llamada a una función que los recoge todos.

AS3: boton.addEventListener ó boton.removeEventListener
 Animate: boton.setEnabled(true) ó boton.setEnabled(false)

```

public void devuelveComportamientos () {
    //enabled =true;
    boton_volver.setEnabled(true);
    botonSiguiente.setEnabled(true);
    menu.setEnabled(true);
    escuchar.setEnabled(true);
    grabar.setEnabled(true);
    cerrar.setEnabled(true);
}

función activaComportamientos(): void {
    botesuchar.addEventListener(MouseEvent.CLICK, Escuchar);
    boton.addEventListener(MouseEvent.CLICK, inicializar);
    if (botgrabar.habilitado) {
        botgrabar.addEventListener(MouseEvent.CLICK,
            onGrabar)
    }
    volver_inicio.addEventListener(MouseEvent.CLICK, voy_inicio);
    button_menu.addEventListener(MouseEvent.CLICK, abrirMenu);
    boton.visible=true;
}

```

65. Función que devuelve los comportamientos de los botones en AS3 y en Java

- 4) Código que lanza las animaciones de mano correcto – error: Ya hemos visto que para lanzar la animación es cuestión se comprueba el resultado obtenido con la palabra que tenemos en el cajetín. Tras comprobar si coincide o no, se lanza una animación u otra.

```

case SPEECH_RECOGNITION_CODE: {
    if (resultCode == RESULT_OK && null != data) {
        ArrayList<String> result =
        data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS);
        boolean correcto = false;
        String palabraActual = palabras.get(numeroTitulo);
        int contador = 0;
        while(contador<result.size() && !correcto){
            if
            (result.get(contador).toLowerCase().equals(palabr
            aActual.toLowerCase())){
                correcto=true;
            }
            contador++;
        }
    }
    if (correcto){
        mano.setImageResource(R.mipmap.mano_bien);
        animacion();
        siguienteFoto();
    }else{
        mano.setImageResource(R.mipmap.mano_error);
        animacion();
        siguienteFoto();
    }
}

var palabritas: Array = jsonParser.parseGoogleSpeechJSON(ev.data);
var resultadoCorrecto: Boolean = false;
for (var i: int = 0; i < palabritas.length; i++) {
    if (palabritas[i] != undefined) {
        if (palabritas[i].toLowerCase() ==
            textoCargado.text.toLowerCase()) {
            resultadoCorrecto = true;
        }
    }
}
if (resultadoCorrecto == true) {
    correcto.play();
} else {
    error.play();
}

```

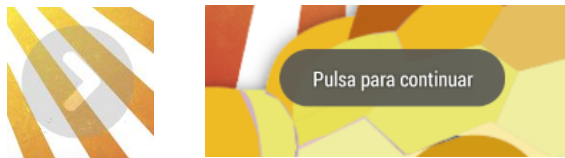
66. Función que lanza la animación correspondiente en AS3 y en Java

5.4. Diferencias

5.4.1. Diferencias en la interfaz

Como hemos visto anteriormente, se aprecian grandes diferencias en las interfaces de ambas aplicaciones. Analizamos cuáles son esas diferencias y su razón de ser:

- **Pantalla de Inicio:**
La pantalla de inicio es, de las tres pantallas que integran la aplicación, la que menos cambios sufre. Cuando el contenido de la aplicación está correctamente cargado en Animate, aparece una flecha en color gris con cierta transparencia que indica que se debe pasar a la pantalla siguiente. Esto se ha cambiado en Android Studio, hemos querido utilizar un elemento característico de Android que hemos explicado anteriormente: el Toast. A modo de recordatorio, un Toast es un elemento que muestra un mensaje durante unos segundos y luego desaparece. De esta manera, al iniciar la aplicación se activa el Toast con el mensaje “Pulsa para continuar”. Este elemento no lo programamos de manera gráfica o desde el archivo xml del activity. Creamos un objeto Toast y hacemos una llamada desde el archivo Java de la pantalla de inicio, donde declaramos el mensaje, la ubicación y el tiempo que se mostrará en pantalla.



67. Gráfico que indica pase de pantalla

Se declara el Toast de la siguiente manera:

```
Toast.makeText(this, R.string.textoMostrar, Toast.LENGTH_LONG).show();
```

Respecto al logotipo de la mano en Animate realiza una pequeña animación de cambio de tamaño, que invita al usuario a tocar sobre éste para realizar el pase a la siguiente pantalla. En Android Studio el funcionamiento es totalmente diferente. Se crea un objeto de tipo Button y se asigna la imagen como “background” a través de la ruta al archivo. La imagen en cuestión ha sido añadida previamente a la carpeta mipmap. Por otra parte, tenemos el archivo de la animación en xml, que realiza el cambio de tamaño en porcentaje sobre el tamaño del objeto. El archivo de la animación se llama blink.xml, donde se declara el porcentaje a aumentar, el modo que en este caso es “repeat” y el número de veces que será infinito. En el archivo java de la pantalla de inicio se crea una función donde se asigna la animación al botón. Se realiza una llamada a dicha función desde la función onCreate de la pantalla de inicio, para que se inicie cuando se abra la aplicación.

Para asignar el pase a la pantalla principal se asigna al botón del logotipo una función que se ejecuta cuando se produzca el clic sobre éste. Se desencadena un intent para empezar la actividad principal y se da la orden de cerrar la actividad actual. Se cierran con ella todos los procesos activos (pase de eslogan y animación del botón).

Por otra parte, la imagen de la cúpula que utilizamos en la interfaz no se carga directamente desde los archivos del programa, pues ocupa mucho espacio y

la aplicación la bloquea. Utilizamos para cargar la imagen un objeto de tipo `ImageView` (lo nombramos `cupula`) y lo situamos en la interfaz, pero no le asignamos ninguna imagen. Desde Java utilizamos la librería Picasso para cargar archivos e imágenes de mayor peso. Nos permite realizar la carga de imágenes desde el servidor, solucionando así numerosos errores de carga y liberando gran cantidad de espacio. Picasso fácil de utilizar, sólo es necesaria una línea de código:

```
Picasso.get().load(https://image.ibb.co/ksuaYc/cupula_interfaz_2.png).into(cupula);
```

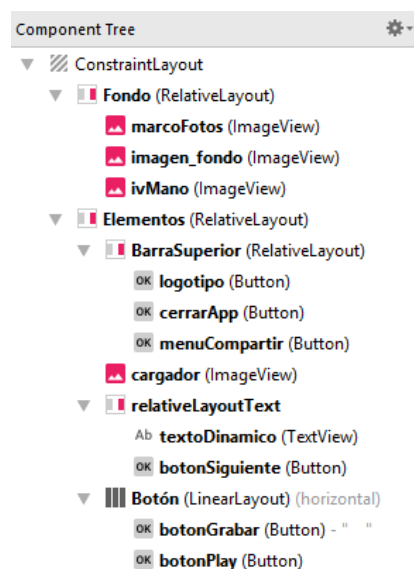
Otro gran cambio que encontramos en Android Studio es el pase de eslóganes:

- A nivel gráfico: El rectángulo que contiene el texto no es un archivo PNG. En Android Studio tenemos la posibilidad de crear formas a través de xml, que nos permite ahorrar memoria y reducir su peso. A través de un archivo llamado `rectangulo_slogan.xml` definimos la forma que es rectangular, especificamos que debe tener las esquinas redondeadas con un radio de 40dp y un color plano del fondo en RGB que es `#a6ffe252`. Asignamos como “background” del cuadro de texto que contiene los eslóganes el archivo xml.
- A nivel de código: En un principio no asignamos ningún texto al cuadro. En java a través de bucle switch declaramos en cada case un eslogan, hasta un total de 10. Creamos un temporizador `CountDownTime` para que cada cinco segundos se ejecute el switch, que cambia el eslogan. Una variable auxiliar almacena el número de eslogan que corresponde que se muestre. Se asigna el texto referenciándolo al cuadro de texto con un método propio llamado `setText(textoEslogan)`. Cada vez que se muestra un eslogan la variable se incrementa en 1. Cuando llega a 10, la variable vuelve a tomar un valor de 1 para que se repitan en bucle.

- **Pantalla Principal:**

La pantalla principal es la que integra el mayor número de cambios con respecto a la aplicación desarrollada en `Animate`. Algunas de estas modificaciones son por eficiencia, ya que esta plataforma nos ofrece realizar operaciones de una manera más eficiente para la ejecución de la aplicación. Otras son a nivel estético o por obligación, al resultar imposible realizar una copia exacta de la que se desarrolla en `Animate`.

En Android Studio utilizamos layouts. Los layout son la manera que tenemos de acomodar los diferentes elementos que componen la interfaz. En nuestro caso, utilizamos un `ConstraintLayout` que, según la versión de Android Studio que utilizamos, viene por defecto. Dentro de éste dividiremos la pantalla en diferentes niveles agrupados en



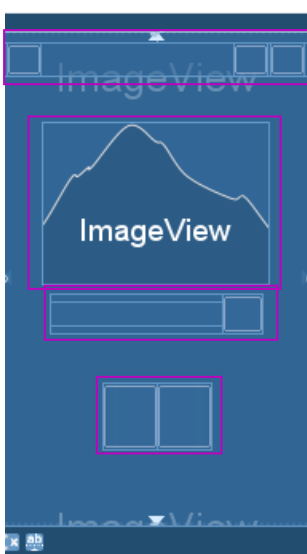
68. Árbol de componentes

los dos grandes bloques que hemos mencionado con anterioridad: fondo y elementos interactivos. A la derecha, podemos ver un esquema del árbol de componentes de los diferentes elementos de la pantalla principal y sus contenedores.

Siguiendo el esquema anterior vemos que tenemos dos RelativeLayout principales: Fondo y Elementos. Ambos se ajustan a las dimensiones de la pantalla. Elementos se encuentra superpuesto al layout Fondo.

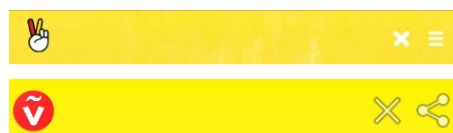
En el Fondo se encuentran los elementos que componen el fondo: el marco de fotos, la imagen de fondo de la Plaza Mayor y también la mano que indica si se ha hecho correctamente el ejercicio o no. La mano se ha programado de una manera diferente a Animate. Se declara un ImageView en la interfaz de la pantalla principal, y no se le asigna ninguna imagen a contener. Se realiza el ejercicio del juego, una vez que se obtiene el resultado se le asigna la imagen correspondiente en cada caso y se lanza la animación con la imagen en cuestión ya cargada. Una vez que se ha completado el tiempo de la animación se cambia la visualización a invisible. Otro elemento que ha sufrido modificaciones en la migración ha sido el degradado del fondo. En Animate el fondo estaba compuesto por un archivo.png de un degradado horizontal de amarillo a blanco. En Android Studio usamos XML para definir estos elementos que puede dibujar el programa, evitando así cargar archivos pesados que ocupen memoria y creen problemas a la hora de mostrarlos en pantalla. Se crea un archivo gradiente_amarillo.xml que contiene las especificaciones del ángulo de gradiente (270°) así como los colores amarillo y blanco en RGB (#FFFF200 y #FFFFFF). Se carga como propiedad background del ConstraintLayout principal.

Dentro de Elementos encontramos diferentes zonas: la barra superior (1), que contiene los botones de avanzar, retroceder y cerrar, el cargador de las imágenes del juego (2), la zona del texto y el botón siguiente (3) y por último el botón grabar-reproducir (4) que se compone a través de un LinearLayout.



70. Zonas de Elementos (RelativeLayout)

- 1) Barra superior: Los iconos de la barra superior muestran una ligera modificación, manteniendo su misma forma y cambiando el aspecto. Se utilizan iconos prediseñados de Android por analogía con otras aplicaciones del sistema operativo. En lugar del utilizar el logotipo de la mano, se usa el icono de inicio pues el usuario ya sabe que es un botón.



69. Barras superiores de ambas aplicaciones

- 2) Cargador de imágenes: es un ImageView vacío en su inicio. Se cargan las imágenes a través de Java utilizando la librería Picasso.

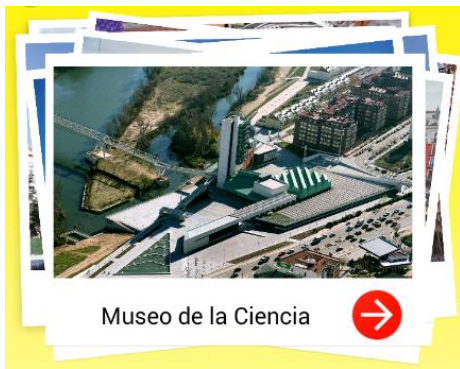


72. Flecha en reposo

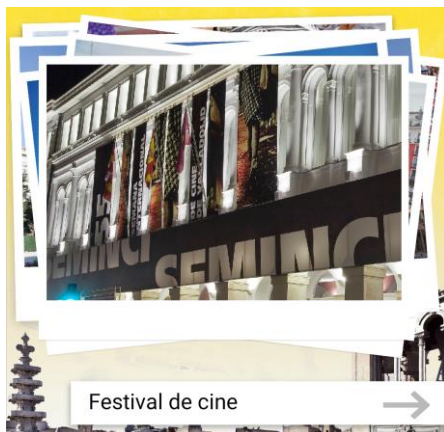


71. Flecha pulsada

- 3) Zona de texto: esta zona cambia totalmente. Por eficiencia, se rediseña este espacio dándole utilidad a la zona inferior del cajetín de las fotografías. Se crea un nuevo icono, en este caso de forma similar al icono de inicio. El tamaño es ligeramente superior para darle protagonismo. Utilizamos dos imágenes para componer el diseño de este botón, que creamos mediante Android Asset Studio. Dichas imágenes son iguales, de forma y de icono, cambiando su fondo tornándose a oscuro. Mediante XML se crean dos ficheros que contienen cada estado del botón. El estado de reposo mostrará la imagen con el color de fondo rojo, mientras que la imagen se oscurecerá cuando se pulse sobre el botón. Esto proporciona un feedback al usuario. Se asigna como background un fichero selector que mostrará en cada caso la imagen correspondiente. Por otra parte, el cuadro de texto desaparece pasando a formar parte del cajetín del marco de las fotografías. El texto se ubica inmediatamente debajo de las fotografías reutilizando elementos y generando más protagonismo para el fondo.



73. Situación de zona de texto en Android Studio



74. Situación de zona de texto en Animate

- 4) Respecto al botón que nos permite reproducir el audio y grabar, está formado por dos botones que se unen a través de un LinearLayout vertical. Cada botón es independiente y su apariencia es similar a los que utilizamos en Animate, pero se componen de forma totalmente diferente. Para realizar el background de cada botón, que son iguales pero opuestos en forma, creamos un archivo XML donde vamos a utilizar el selector que utilizamos en el botón siguiente. En este caso, en vez de seleccionar archivos.png, definimos cada estado dentro del XML. Cada estado se compone de forma, color gradiente, forma de las esquinas, sombra, color de la sombra y su forma. En el estado presionado el color de fondo pasa de gradiente a color plano ligeramente más oscuro. Ambos botones, cada uno con su archivo XML que contiene la configuración del fondo, forman parte del LinearLayout que se ajusta a sus dimensiones. Este contenedor los mantiene unidos, y dada su apariencia, simulan un solo botón. El LinearLayout pertenece al contenedor superior (RelativeLayout - Elementos) que lo contiene y posiciona en la parte centro-inferior de la pantalla.

En la siguiente página se adjunta el ejemplo de un estado, para comprenderlo mejor.

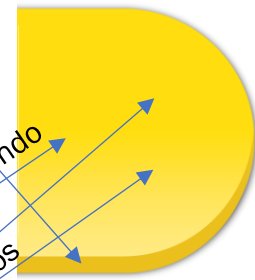
```

<?xml version="1.0" encoding="utf-8"?>
<selector
xmlns:android="http://schemas.android.com/apk/res/an
droid">
  <!-- Boton Reposo-->
  <item android:state_pressed="false">
    <layer-list>
      <!-- Sombra -->
      <item>
        <shape>
          <solid android:color="#d6b510"/>
          <corners
            android:topRightRadius="40dp"
            android:bottomRightRadius="40dp"/>
        </shape>
      </item>
      <!-- Botón-->
      <item
        android:bottom="10px"
        >
        <shape>
          <gradient
            android:endColor="#ffe45a"
            android:startColor="#ffd814"
            android:angle="270" />
          <padding
            android:bottom="10dp"
            android:left="10dp"
            android:right="10dp"
            android:top="10dp"
            >></padding>
          <corners
            android:topRightRadius="40dp"

```

Desplazamiento de la forma principal respecto de la parte inferior

Gradiente de fondo
Radios



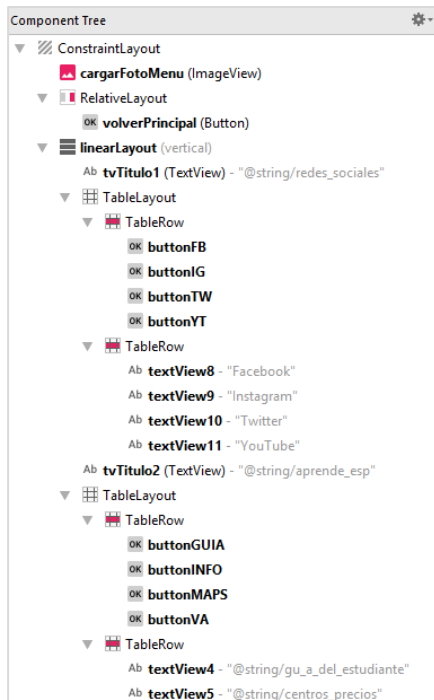
75. Relación entre archivo XML e interfaz

- Menú de desbordamiento:

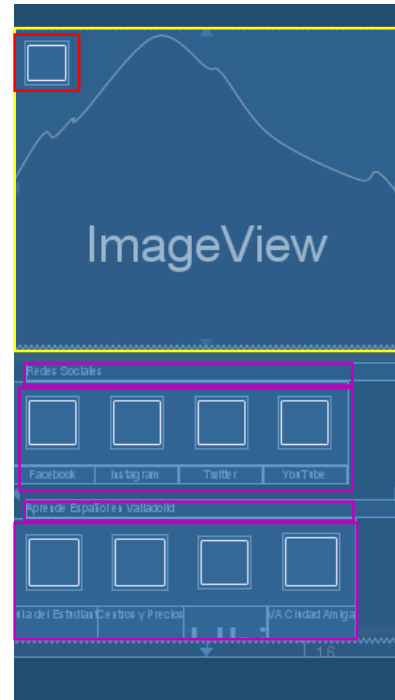
El menú de desbordamiento cambia totalmente de apariencia. Pasa de ser un menú que emerge de la parte inferior de la pantalla, a aparecer desde el lateral derecho desplazándose hacia la izquierda. Es una actividad nueva, que al iniciarse carga una fotografía utilizando la librería ya mencionada Picasso. Debajo de la foto, que aporta luz y protagonismo al menú, encontramos los botones a los distintos enlaces que forman dicho menú. En este caso, se encuentran distribuidos en dos contenedores horizontales, donde encontramos el grupo de enlaces de Redes Sociales y el de Aprende español en Valladolid. A continuación, se muestra el árbol de componentes del menú junto con su distribución en pantalla.



76. Menú de desbordamiento en Android Studio



77. Árbol de componentes



78. Grupos que contiene el ConstraintLayout

Dentro del contenedor principal, que es el ConstraintLayout, se ubican el ImageView donde se carga la foto principal del menú: cargarFotoMenu, un RelativeLayout que nos permite colocar el botón de volver y el LinearLayout horizontal sobre el que se organizan los botones y textos de la actividad. El ImageView que carga la foto ocupa un poco más de un tercio de la pantalla. Se ha introducido esta modificación porque el menú de los enlaces, a lo largo del proceso de diseño, el menú queda un poco frío y oscuro. De esta manera invita a querer conocer más sobre el programa. El RelativeLayout se adapta a las dimensiones de la pantalla. Encontramos el botón situado arriba a la izquierda y superpuesto a la foto de fondo, que es el que nos permite volver a la actividad principal. Se utiliza como imagen una flecha prediseñada por Android Studio. Se añade este botón que no existe físicamente en la otra versión de la aplicación, ya que en Animate tocando sobre el fondo se volvía a la pantalla principal y en este caso esa opción no era posible.

Se utiliza un LinearLayout para la disposición de los botones y los enunciados porque es aquel contenedor que mejor permite adaptar el contenido a la pantalla. Dentro de éste, encontramos los títulos de los dos grandes bloques junto con dos contenedores TableLayout. Cada contenedor TableLayout integra dos filas: la primera con los botones de cada grupo y la segunda con las etiquetas que corresponden a cada botón. Cada botón tiene una imagen.png asignada en sus propiedades background y una función que enlaza con la página web en cuestión.

5.4.2. Diferencias en el código

El código de ambas aplicaciones está desarrollado en lenguajes de programación diferentes, de acuerdo con el entorno de desarrollo en cuestión. Podemos señalar algunas diferencias significativas entre una aplicación y otra.

- Navegación entre escenas:
Dedicaremos el siguiente apartado a hablar sobre la navegación en la aplicación. En Android Studio se crean funciones y objetos específicos para permitir esa navegación entre las actividades.
- Carga de los elementos de la pantalla principal:
Como hemos explicado en el apartado de desarrollo de la aplicación en ActionScript 3.0, en la pantalla de inicio se cargan los componentes de la pantalla principal. En Android Studio debemos extraer los datos de las palabras y las posiciones del audio antes activar el comportamiento de los botones. Se utilizan otros métodos para realizar las mismas acciones. Se utiliza la clase final `AsyncTask`, que integra una serie de métodos que se ejecutan en el hilo correspondiente para realizar cualquier tarea que comprometa el funcionamiento o pueda bloquear la interfaz. Los métodos que utilizamos para realizar las funciones de carga de los componentes de la pantalla principal son:
 - `doInBackground()`: la tarea a ejecutar se realiza en un hilo (`Thread`) diferente para evitar bloquear la interfaz.
 - `onPostExecute()`: se realiza la llamada a este método cuando se termina el método anterior. Aquí es donde introducimos las acciones a realizar una vez que se cargan los componentes.

Utilizamos dos clases `AsyncTask`, en la primera cargamos las palabras y en la segunda las posiciones de cada audio.

(1) `final class CargaPalabras extends AsyncTask`

- `doInBackground()`: Se le indica que debe leer el archivo que se le pasa en la llamada a través de la dirección web. Lee el archivo de texto línea por línea y cada una de ellas la almacena en un `Arraylist`. Se utiliza `Arraylist` en lugar de `Array` ya que carece de dimensión fija. En caso de modificar el archivo, introduciendo o quitando datos, no es necesario modificar el código de la aplicación.
- `onPostExecute()`: Una vez que tenemos las palabras que se corresponden con las fotos, se carga la primera imagen junto con el texto correspondiente. La correspondencia del índice de las fotografías y el de las palabras sigue la misma lógica en ambas aplicaciones. Se carga la fotografía a través de `Picasso`, componiendo la URL de esta a través de diferentes `String`:

```
String imagen = "http://diseno.eii.uva.es/~multim14/app/imagen";  
String jpg = ".JPG";  
Picasso.get().load(imagen+ fotoMostrada + jpg).into(cargarFoto);
```

(2) `final class cargaLimites extends AsyncTask`

- `doInBackground()`: Se lee el fichero donde están contenidos los límites de los audios y se almacenan en un `Arraylist` llamado

límitesPalabras, siguiendo la misma lógica que en el método anterior.

- `onPostExecute()`: Se realiza la llamada a una función que extrae, de cada línea almacenada en el `Arraylist` anterior, las posiciones de comienzo y duración de cada palabra, que se encuentran separados por un tabulador(`\t`), y las almacena en dos `Arraylist` distintos: `audioComienzo` y `audioDuracion`. Ambos `Arraylist` deben de ser tipo `Float`, para almacenar los decimales de cada instante. Se indica al reproductor de sonido, creado anteriormente, la URL donde se encuentra el archivo de sonido y se deja listo para reproducir el sonido cuando se indiquen los parámetros necesarios.

- Animación de mano correcto/error:
Hemos visto en las diferencias de la interfaz que para programar el funcionamiento de la animación de la mano se define un elemento tipo `ImageView`, que en su inicio está vacío. Tras realizar la valoración del ejercicio se realiza un bucle `if else` en función del resultado del ejercicio. Dependiendo que parte del bucle se realice se carga una imagen u otra en el `ImageView`. Después se realiza la llamada a una función denominada `animacion()`.

Esta función lo primero que hace es activar la visibilidad del `ImageView` de la mano. Mediante un nuevo objeto de tipo `ScaleAnimation` se define la animación que va a realizar la mano, un cambio de tamaño desde el centro conservando sus proporciones. Se indica la duración de la animación en milisegundos (420), el modo (`Reverse`) y el número de repeticiones(3). Se inicia la animación y se declara un `Handler` para que pasado el tiempo establecido de duración de la animación, la mano cambie su visibilidad a invisible. En otro apartado hemos explicado una manera diferente de programar una animación definiéndola a través de un archivo XML.

The image displays three components related to Android animation:

- XML Code:**

```
<?xml version="1.0" encoding="utf-8" ?>
<set xmlns:android="http://schemas.android.com/apk/res/a
  <scale
    android:fromXScale="1"
    android:fromYScale="1"
    android:pivotX="50%"
    android:pivotY="50%"
    android:toXScale="1.2"
    android:toYScale="1.2"
    android:interpolator="@android:anim/accelerate_interpolator"
    android:duration="600"
    android:repeatMode="reverse"
    android:repeatCount="infinite"
  >
</scale>

  <alpha android:fromAlpha="0.8"
    android:toAlpha="1.0"
    android:interpolator="@android:anim/accelerate_interpolator"
    android:duration="600"
    android:repeatMode="reverse"
  >
</alpha>
```
- Java Code:**

```
scale=new ScaleAnimation(
  1, 2.5f, 1, 2.5f,
  Animation.RELATIVE_TO_SELF
  , 0.5f,
  Animation.RELATIVE_TO_SELF
  , 0.5f);
scale.setDuration(420);
scale.setRepeatMode(Animat
```
- Timeline:** A visual representation of the animation sequence. It shows a horizontal axis with markers at 1, 5, 10, 15, 20, 25, 30, 35, and 40. Two tracks are visible: 'reposo' (rest) and 'escuchando' (listening). The 'reposo' track shows a blue bar starting at 0 and ending at 20. The 'escuchando' track shows a blue bar starting at 20 and ending at 40. A red vertical line is positioned at the 20 mark, indicating the transition point between the two states.

79. Definición de la animación en XML, Java y línea de tiempo de `Animate`

- Reconocedor de voz:

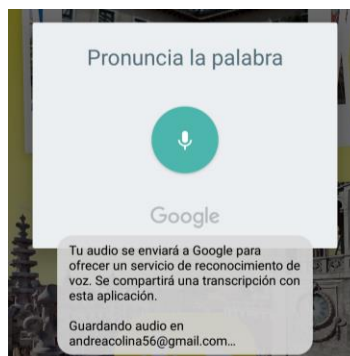
A la hora de transcribir las palabras pronunciadas por el usuario hacemos uso del servicio reconocedor de voz de Google. Ya hemos explicado anteriormente el servicio de pago Google Speech API que utilizamos en Adobe Animate. Animate carece de las librerías nativas de las que cuales disponemos en terminales Android. Es por ello por lo que necesitamos hacer uso del servicio de pago. En Animate utilizamos el reconocedor de Google de pago, por ello hacemos uso del archivo de configuración que nos permite inhabilitarlo, así como guardar la contraseña del servicio contratado. Utilizamos este servicio porque la aplicación es un proyecto que requiere ser exportado a diferentes plataformas (iOS, Windows Phone, ...) y la entidad ha destinado una parte del presupuesto al uso de este recurso.

Por el contrario, al implementar una aplicación que se desarrolla para terminales Android, Google nos permite hacer uso de sus servicios de manera gratuita. No es así si requerimos guardar el audio, pero en nuestro caso no será necesario. Utilizando las clases nativas, activamos el reconocedor de voz pulsando el botón de grabar. Al pulsar el botón se hace uso de una función llamada `onRecordClick` que recoge el evento de pulsar sobre el botón grabar. En el interior de esta función está situada una llamada a la función `startSpeechToText()` que es la que nos permite hacer uso del reconocedor. Para utilizar esta función de Google tenemos dos funciones clave en el código Java de la aplicación:

(1) **private void startSpeechToText():**

Esta función nos permite especificar una serie de parámetros para definir el audio como, por ejemplo: el lenguaje del audio, que es español del España (es-ES) o el número de resultado posibles (5) que deseamos obtener del ejercicio.

En este punto no tenemos ningún acceso a los procesos de Google de transcripción del audio. Al utilizar el reconocedor se muestra una interfaz propia del servicio de Google.



80. Interfaz del reconocedor de Google

(2) **protected void** onActivityResult():

Es la función que recoge el resultado del ejercicio. En primer lugar, se comprueba que la grabación ha sido favorable y que hay datos que cotejar. En segundo lugar, se transfieren las cinco posibilidades que se recogen como transcripción y se almacenan en un ArrayList. Después haciendo uso de un bucle do-while se comprueba si la palabra a pronunciar coincide con alguna de las posibilidades. Una variable boolean almacena el resultado de esta operación. En función de ese resultado se lanza, como ya hemos explicado anteriormente, la animación con la de imagen de una mano u otra.

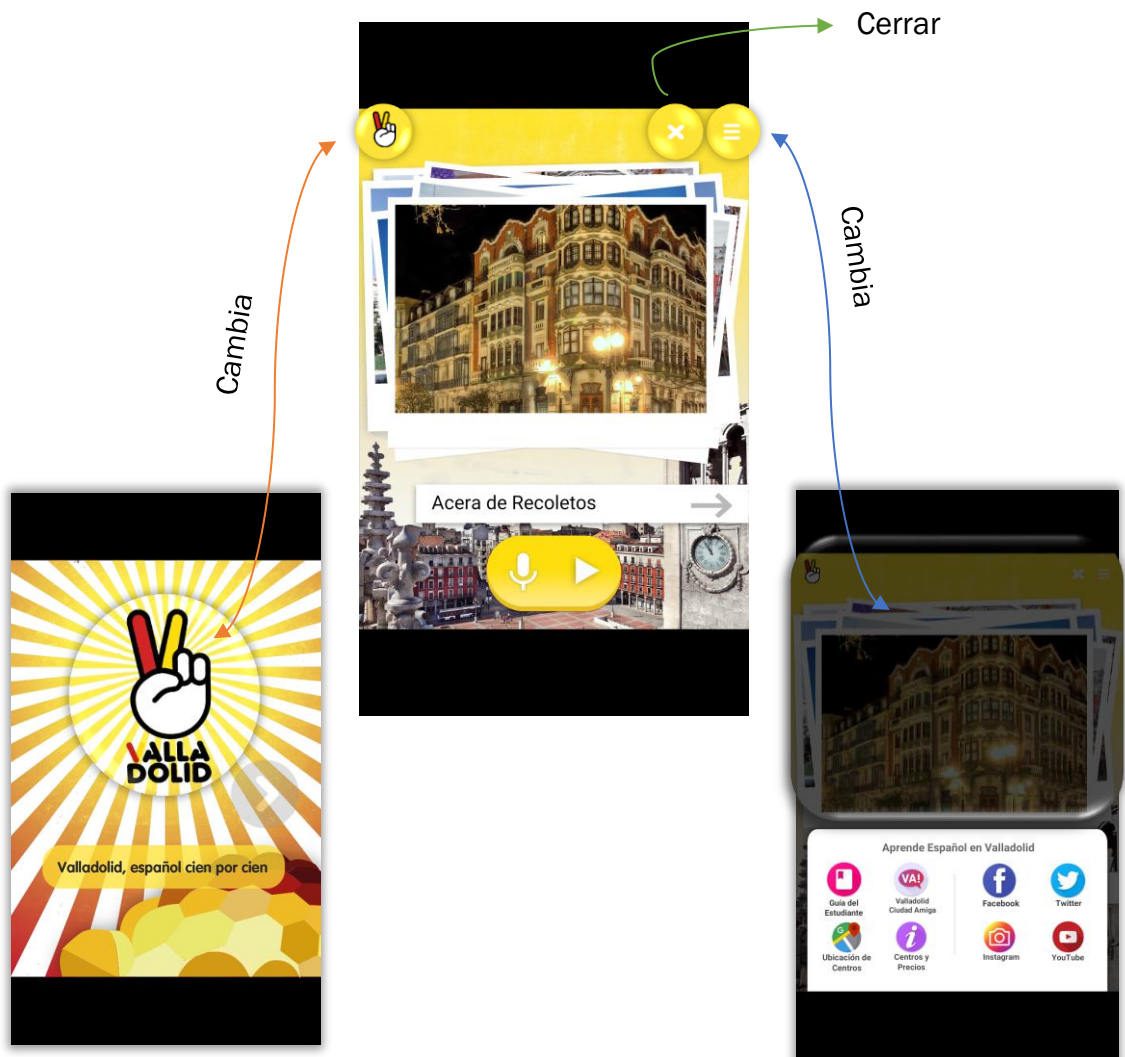
- Reproductor de sonido: Como hemos mencionado anteriormente se declara en la clase principal un objeto de tipo MediaPlayer llamado reproductor. Creamos una función llamada reproducirSonido(). Esta función se lanzará automáticamente al realizar el pase a la siguiente fotografía o al pulsar sobre el botón de reproducir.

Para utilizar el reproductor en primer lugar extraemos los instantes de tiempo de inicio y duración. De la misma manera que en Animate, se encuentra en segundos y necesitamos milisegundos, por tanto, multiplicamos por 1000. Por las características del reproductor y el Handler que vamos a utilizar después, pasamos el instante de inicio a Int y el de fin a Long. A través del método seekTo(inicio) posicionamos el reproductor en el instante de inicio. Lo iniciamos, desactivamos los comportamientos de los botones y declaramos un Handler indicando que, pasado el tiempo de duración de la palabra, debe parar el reproductor y devolver los comportamientos a los botones.

5.4.3. Diferencias en la navegación entre pantallas

Como hemos podido ver a lo largo de todo el desarrollo, cada entorno de desarrollo permite hacer las cosas de una manera u otra. Se busca dentro cada uno, la manera más eficiente de desarrollar la aplicación. En este sentido, no encontramos puntos en común en el funcionamiento de ambas, pese a que las dos realicen las mismas acciones. A modo de resumen de la navegación entre pantallas podemos decir:

- **Animate:** La escena principal de la aplicación es la pantalla principal. Sobre ella se ocultan o aparecen las otras dos escenas, que están contenidas en movieclips. Al iniciar la aplicación, el movieclip de la pantalla de inicio aparece visible ocupando la totalidad de la pantalla. Al hacer clic sobre él, se oculta dejando ver la pantalla principal que se encuentra detrás de éste. Del mismo modo, al pulsar sobre el icono que nos lleva al menú de desbordamiento, lo hacemos visible e iniciamos su animación. Si retrocedemos pulsando sobre la pantalla, ocultamos su vista. La pantalla principal siempre se encuentra presente, oculta o no, según en momento, por los elementos que sitúan sobre ella.



81. Gráfico de navegación entre pantallas en Animate

- Android Studio: Cada pantalla está definida en una actividad diferente. A través del archivo Manifest se declara que tiene que ser la pantalla de inicio la que debe aparecer al abrir la aplicación. A cada botón se le asigna un callback onClick(), de manera que cuando la clase View recoja el evento que se produce sobre dicho botón se desencadenará la función contenida en onClick. Por ejemplo:

```
public void goToFacebook (View view) {
    goToUrl ( "https://www.facebook.com/learnSpanishinV11");
}
```

Definimos las funciones en Java. No necesitamos declarar el setOnClickListener desde el archivo Java, lo hacemos desde XML. En este archivo se definen las propiedades de cada botón. Dentro de esas propiedades añadimos:

```
<Button
    android:id="@+id/buttonFB"
    android:layout_width="56dp"
    android:layout_height="56dp"
    android:layout_margin="16dp"
    android:background="@mipmap/ic_facebook"
    android:onClick="goToFacebook" />
```

En cuanto a la navegación entre escenas, utilizamos un objeto llamado Intent. Un intent describe la actividad que se debe iniciar y contiene las instrucciones para iniciarla.

Por ejemplo, al pulsar sobre el botón que contiene el logotipo, se desencadena la función pasarPantalla() donde se crea un objeto de tipo Intent al cuál se le indica dónde está y a dónde tiene que ir. A través del método startActivity(intent) se inicia la nueva instancia. Por último, se ordena cerrar la actividad actual.

```
public void pasarPantalla(View Vprincipal){
    Intent intentInicio= new
    Intent(PantallaInicio.this,PantallaPrincipal.class);
    startActivity(intentInicio);
    this.finish();
}
```



82. Gráfico de navegación entre pantallas en Android Studio

6. Usabilidad

¿Qué es? La usabilidad es la facilidad que tiene una persona para realizar una acción o interactuar con una herramienta con el fin de conseguir un objetivo.

En este sentido, la estructura de la aplicación debe ser cuidada y estudiada debidamente. Se deben tener en cuenta los gestos que realiza el usuario, el número de manos que se utilizan y la posición del dispositivo. En nuestro caso la disposición del dispositivo es vertical, por tanto, debemos tener en cuenta las zonas de interacción de los dedos y el acceso a ellas.

El contenido de la aplicación es de vital importancia para el usuario, pasando a situarse el diseño de la interfaz en un segundo plano. El usuario espera que la información se muestre de una manera clara y directa. Las imágenes deben ser grandes y tienen un peso importante para éste, mientras que el texto debe ser claro y conciso, ya que el espacio para mostrarlo es reducido. Podemos decir que una aplicación es una versión reducida, básica, de una página web. Una aplicación nunca será más completa que la versión web de la misma.

Se sobreentiende que la aplicación debe estar optimizada para todo tipo de dispositivos, tamaños y sistemas operativos. Respecto al proyecto que se desarrolla para el Ayuntamiento, el que se desarrolla con Adobe Animate, está disponible para diversas plataformas: Android x86 y ARM, WindowsPhone, iOS y ejecutable para Windows.

La usabilidad de los dispositivos móviles se plantea como un problema real, ya que a menudo las aplicaciones son difíciles de utilizar. Como consecuencia de la movilidad de los dispositivos, las pruebas de usabilidad entrañan una dificultad bastante grande.

Como pruebas de testeo de usabilidad se comprueban las reacciones de los usuarios al utilizar por primera vez la misma. Se escoge un grupo reducido de personas (5 personas) y se realiza una observación de campo. Se recogen las reacciones de cada usuario frente a las interacciones con la aplicación. También se obtienen datos a través de entrevista, escuchando opiniones y valoraciones.

- Se observa excesiva rapidez por pasar a la pantalla principal, tocando repetidamente sobre el icono hasta que se consigue visualizar la pantalla principal.
- En la pantalla principal hemos comprobado que el gesto que por inercia que utiliza el usuario para pasar de fotografía es deslizar el dedo sobre la que se está mostrando. Este gesto es heredado del funcionamiento de los sistemas operativos y otras aplicaciones.
- También se observa que durante el tiempo que reconocedor está esperando la respuesta, el usuario se encuentra expectativo ante que pasará. Quizás se debe recalcar el mensaje de “esperando respuesta” para que el usuario sea consciente mucho antes.
- Respecto al menú de desbordamiento, el usuario siente desconcierto a la hora de cerrarlo, bien mediante el botón físico de “volver” o tocando sobre la zona negra.

Estos comportamientos deben ser valorados en la medida en que sirvan para realizar mejoras en la aplicación de cara a futuras versiones de la aplicación, pues todo debe ser susceptible de cambio. Concretamente, deben ser analizados de cara a realizar

mejoras en la aplicación existente publicada en Google Play. Recordemos que las aplicaciones están en una constante mejora y evolución.

7. Conclusiones

Una vez que se da por finalizada la fase de desarrollo de ambas aplicaciones se puede concluir que se han alcanzado con creces los objetivos y especificaciones que se marcaron al inicio del proyecto.

Se realiza el rediseño de la interfaz de la aplicación existente. Se le confiere una nueva imagen, que va acorde con el programa Aprende español en Valladolid, pasando por un proceso de bocetaje y selección de propuestas. A nivel personal, esta experiencia supone el primer contacto con el mundo laboral. Al trabajar de cara a un cliente, se establecen fases y plazos de entrega que se cumplen holgadamente. La implementación de la aplicación se realiza con Adobe Animate. Se afianzan los conocimientos tanto de este programa como del lenguaje de programación ActionScript 3.0. Se adquiere gran soltura y destreza en el entorno de desarrollo en cuestión. También se realiza la exportación del ejecutable de la aplicación a los principales sistemas operativos que lideran el mercado (Android, iOS, WindowsPhone).

La migración es un proceso que requiere una rápida adaptación al entorno de desarrollo en el que estamos trabajando en cada momento. Se plantean retos de manera abierta y se resuelven, en la mayoría de los casos, de forma distinta. Esto muestra riqueza de conocimiento y nivel resolutivo a la vez que permite comprobar la eficiencia y eficacia de la aplicación desarrollada de una manera u otra.

Se decide realizar la migración al entorno de desarrollo de Android Studio. Ello implica, en primer lugar, invertir meses de formación y preparación en el lenguaje de programación Java. La decisión de implementar la aplicación en un lenguaje de programación u otro puede tener grandes impactos de cara al futuro. Java es mundialmente popular, además se encuentra apoyado por unos de los sistemas operativos más famosos del mundo: Android. De cara a lo personal, Java es una de las 20 habilidades más valoradas según LinkedIn, de manera que se busca una formación con utilidad de cara al futuro. En segundo lugar se adquiere formación sobre el lenguaje de marcado XML, que utilizamos para definir los elementos que componen la interfaz. En tercer lugar se realiza un estudio acerca de las guías de diseño de Android, conocidas como Material Design, que nos permiten dotar a la aplicación de facilidad de uso. También se realizan múltiples aplicaciones a modo de prueba, que complementarán los ensayos de la aplicación final. Todos estos conocimientos, que en su mayoría se adquieren de manera autodidacta, se plasman en la migración de la aplicación en Android Studio, que se realiza con éxito y superan con creces las expectativas iniciales.

Se aprende a utilizar el reconocedor de audio bien utilizando aplicaciones nativas del sistema Android o bien como un servicio exterior que ofrece Google y que integramos en nuestra aplicación.

Como experiencia personal ha sido muy enriquecedora tanto a nivel de conocimientos adquiridos y reforzados como a nivel de desarrollo de un proyecto y experiencia laboral. He aprendido a trabajar en grupo y responder a las demandas de un cliente así como he podido poner en práctica los conocimientos aprendidos a lo largo de los años de grado y esclarecer hacia donde quiero orientar mi vida laboral.

8. Bibliografía

[1] Adobe Flash Platform, ActionScript 3.0 Developer's Guide

https://help.adobe.com/en_US/as3/dev/index.html

[2] Android Studio Faqs, Cambiar el color de fondo del Layout

[https://androidstudiofaqs.com/tutoriales/cambiar-color-de-fondo-de-layout-en-android-studio\(Noviembre 2015\)](https://androidstudiofaqs.com/tutoriales/cambiar-color-de-fondo-de-layout-en-android-studio(Noviembre 2015))

[3] Ayuntamiento de Valladolid, Manual de identidad visual y aplicaciones gráficas "VA! Valladolid Ciudad Amiga"

<https://www.valladolid.es/es/ayuntamiento/organizacion-administrativa/alcaldia/hacemos/identidad-corporativa>

[4] Ayuntamiento de Valladolid, Página web Aprende español

<http://www.info.valladolid.es/aprende-espanol>

[5] Beginners Book – Java Tutorial: Learn Java Programming with examples

<https://beginnersbook.com/java-tutorial-for-beginners-with-examples/>(Agosto 2017)

[6] Downey, A. B., & Mayfield, C. (2016). Think Java: How to Think Like a Computer Scientist. " O'Reilly Media, Inc."

[7] Calvo, Jesús. Curso de Java para principiantes: Introducción y fundamentos de la programación

<http://www.cursopedia.com/Ficha-Curso-de-Java-para-principiantes>

[8] Cuello, J., & Vittone, J. (2013). Diseñando apps para móviles.

[9] Curso Primeros Pasos com Android – Código Facilito

<https://codigofacilito.com/cursos/android-principiantes>

[10] Departamento de Ciencia Computacional, Drawables, estilos y temas

<http://www.jtech.ua.es/dadm/restringido/android/sesion05-apuntes.html#Colores>(Diciembre 2012)

[11] Ejercicios básicos de programación en Java

<http://puntocomnoesunlenguaje.blogspot.com.es/p/ejercicios.html>(Noviembre 2013)

[12] Ejercicios propuestos y resueltos de Java – Disco Duro de Roer

www.discoduroderoer.es/category/ejercicio/ejercicios-java/web/

[13] Escudero-Mancebo, D., E. Cámara-Arenas, E., Tejedor-García, C., González-Ferreras, y V. Cardeñoso-Payo. “Implementation and Test of a Serious Based on Minimal Pairs for Pronunciation Training”. *Proceedings of SLaTE 2015*. Prensa. (marzo 2017)

[14] Mancebo, D. E. (2003). Fundamentos de informática gráfica. Cano Pina.

[15] F.Lantigua, Isabel. El móvil supera por primera vez al ordenador para acceder a Internet. El Mundo

<http://www.elmundo.es/sociedad/2016/04/04/57026219e2704e90048b465e.html> (Abril 2016)

[16] Google, Guía de Material Design

<https://material.io/design/>

[17] Hermosa Programación, “Como crear tu primera aplicación Android desde cero”

[18] Interfaz de usuario en Android: Layouts

<http://www.sgoliver.net/blog/interfaz-de-usuario-en-android-layouts/> (Agosto 2010)

[19] Iván López Montalbán, Manuel Martínez Carbonell y Juan Carlos Manrique Hernández, Android. (2015) Programación multimedia y dispositivos móviles.

[20] Jakuben, Ben. Java Basics for Android Development (Part 1 y 2)

<http://blog.teamtreehouse.com/java-basics-for-android-development-part-1/>(Marzo 2013)

[21] Nurik,Roman. Android Asset Studio

<https://romannurik.github.io/AndroidAssetStudio/index.html>

[22] Sánchez, J.M. ¿Cambiará el iPhone 7 la apatía de los españoles hacia los iPhones? Diario ABC

https://www.abc.es/tecnologia/moviles/telefonía/abci-cambiara-iphone-7-apatia-espanoles-hacia-iphones-201609062146_noticia.html(Septiembre 2017)

[23] StackOverFlow users, Resolución de dudas

<https://stackoverflow.com/questions/>

[24] Schildt, H. (2006). Java: A Beginner's Guide. McGraw-Hill, Inc..)

[25] Sociedad Mixta Para La Promoción Del Turismo De Valladolid, Folletos Aprende español en Valladolid

<http://www.info.valladolid.es/documents/20147/0/Aprende%20Español%20en%20Valladolid/154fbd67-ece5-0534-4ac3-73ca7767f41d?version=1.0>

[26] Portero López, Violeta Yolanda. Trabajo de fin de Grado: Videojuego educativo para el aprendizaje de español. (Julio 2017)

[27] Zamora, Jose Ángel. La revolución de las interfaces con la creación de ConstraintLayout.

<https://elandroidelibre.elespanol.com/2016/10/revolucion-interfaces-android-constraintlayout.html> (Febrero 2016)