

Memoria Técnica



Aplicación controlada mediante gestos y asistente virtual para un museo

Autores: Antonio Jesús Heredia Castillo
Jose Manuel Perez Lendinez

Resume:	2
Diseño	3
Librerías usadas para la app	7
Picasso	7
QRCodeScanner	7
Firebase Database	7
SimpleFingerGestures	7
Sensores utilizados	8
Cámara-Lector de Qr	8
NFC	10
Multitouch y gestos	11
Acelerómetro	13
Proximidad	14
Implementación de Dialog Flow	14
Dialog Flow	14
Almacenamiento de la información.	14
vector de salas	15
Vector de cuadros	15
Entities	15
Intents	17
Guía	17
Información sobre Salas	23
Listar los cuadros de una sala.	24
Listar las salas con las colección que contiene.	24
Descripción de una sala	25
Información sobre colecciones	26
Descripción de una colección	26
Obtene obras de una colección	26
Información de obras	27
Descripción, sala, colección y descripción obra específica	27
Descripción, sala, colección y descripción obra anterior	29
Nombre de la obra actual.	30
Listar obras de un autor	30
Información completa obra	31
Referencias	35

Resume:

El objetivo de este proyecto ha sido realizar una app para móviles que tuviera funciones de guía en un museo. Para ello se ponían como objetivo integrar nuevos paradigmas de interacción Humano-Máquina.

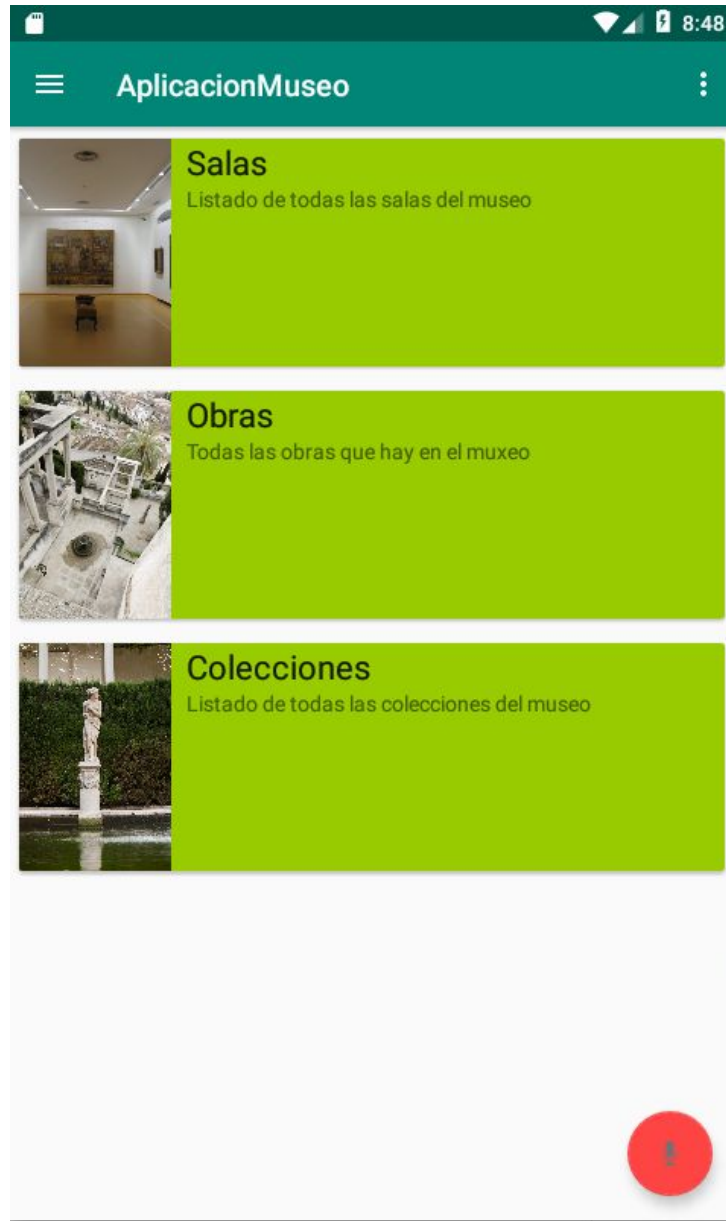
Una de las bases del proyecto ha sido integrar un asistente virtual, que de forma oral diera indicaciones del museo, información sobre las obras y sirviera como guía. Esta opción se podrá usar tanto en la app, como a través de las funciones del asistente de google.

El otro pilar del proyecto es usar los diferentes sensores del móvil. Para ello usaremos sensores como la cámara, el nfc, el giroscopio y el multitouch de la pantalla. Estos sensores darán nuevas funcionalidades dentro de la app para que al usuario le sea más fácil el uso de la misma.

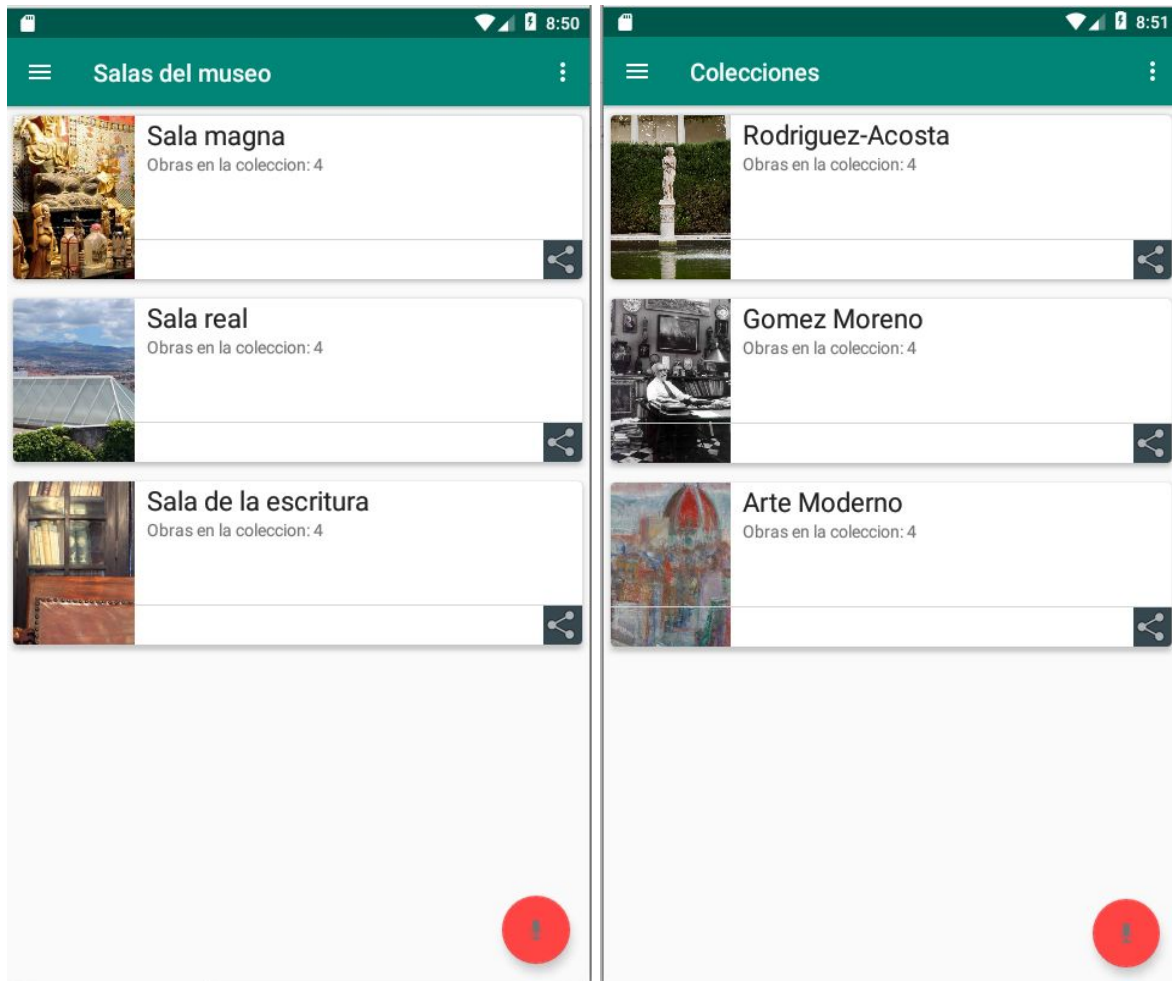
El back-end la aplicación contará con una base de datos en tiempo real de tipo NOSQL y la api de DialogFlow de Google para la parte de asistente conversacional.

Diseño

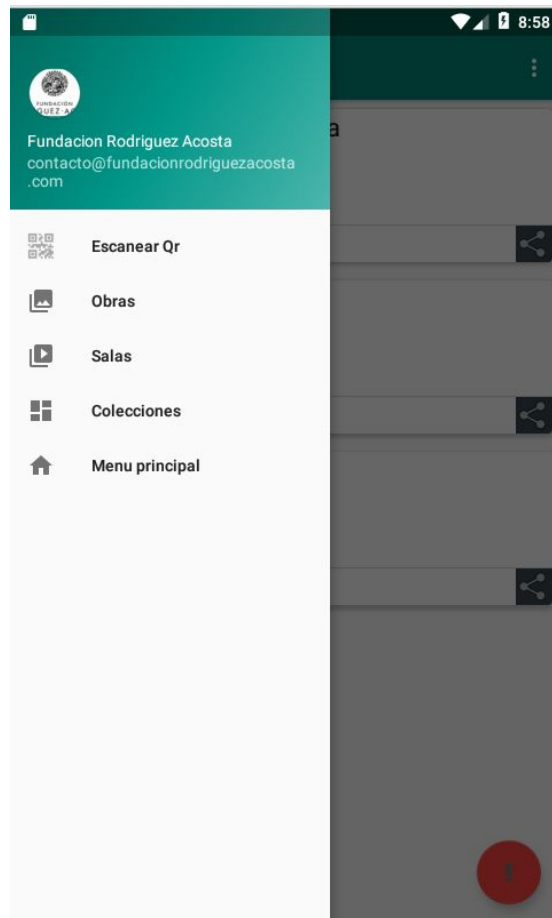
Para el diseño de la de la app hemos elegido vistas simples, con poca información para que el uso de la misma sea fácil e intuitiva.



Para las diferentes activity, hemos elegido diseños iguales, para que haya coherencia entre las diferentes pantallas.



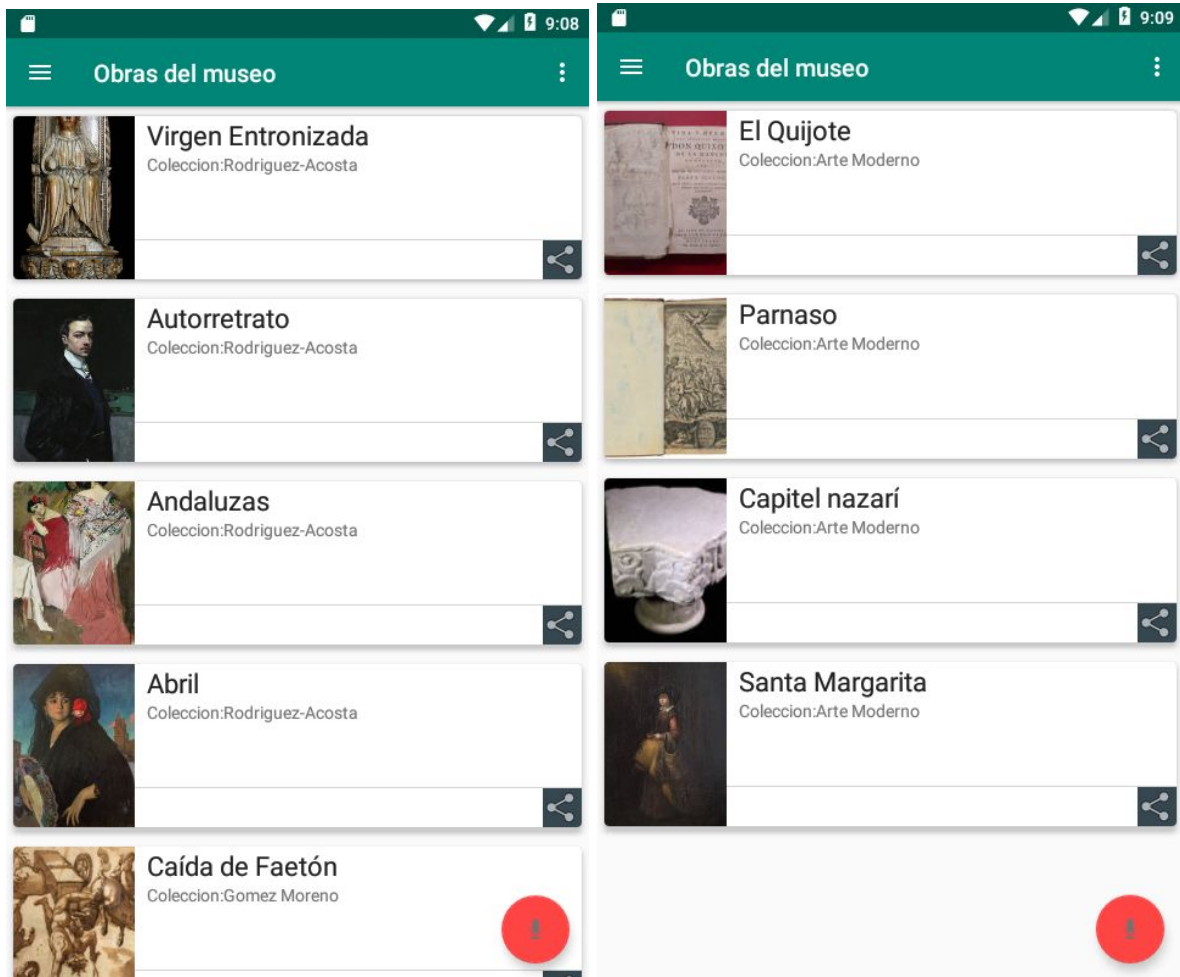
Para cambiar más fácil entre lo que quiere ver el usuario, hemos añadido un menú lateral, para tener un fácil acceso a las distintas opciones de nuestra aplicación.



En la activity, de mostrar la información sobre una obra se muestra la descripción de la misma, la imagen y el título es el nombre de la obra.



En todas la activity se incluye un botón flotante para iniciar el asistente de voz virtual. La activity que muestra las obras, mostra diferentes obras según desde donde se lance la activiity, es decir, si por ejemplo entramos desde el menú lateral, mostrará todas las obras existentes en el museo. En cambio si entramos desde desde una etiqueta NFC o Qr, nos mostrará solo las obras de esa sala o de esa colección.



Librerías usadas para la app

Para la creación de la aplicación móvil hemos usado diferentes librerías, que nos proveen de herramientas para facilitar la creación de la misma.

Picasso

Es una librería Open Source, que facilita la descarga de imágenes en la aplicación para mostrarlas en la app en un `ImageView`. Además mostrarlas las guarda en cache para que las próximas veces que se muestre se haga de forma más rápida y con menos carga para la red. Hemos elegido esta opción para que la aplicación sea de un tamaño reducido ahorrando así memoria al usuario. La librería proporciona métodos para añadir imágenes al `ImageView` a partir de una URL.

QRCodeScanner

Es una librería basada en `zxing` para dispositivos android con una API 15 o superior. Esta librería nos facilita el reconocimiento de códigos Qr en la aplicación. Esta librería nos provee de una actividad que lanza la cámara y al reconocer el Qr, vuelve a la actividad desde la que se lanzó y en el `onActivityResult()` podemos obtener la información que contenía el Qr.

Firebase Database

Firebase Realtime Database es una base de datos alojada en la nube. Los datos se almacenan en formato JSON y se sincronizan en tiempo real con cada cliente conectado. Hemos elegido usar esta base de datos en nuestro proyecto para que los datos no estén escritos de forma persistente en el código de la aplicación. Los datos se reciben de forma asíncrona en la app, y cuando se cambian en la base de datos también se cambian de forma automática en la aplicación.

SimpleFingerGestures

SimpleFingerGestures es una librería que nos ayuda a detectar diferentes gestos en la app y cuantos dedos se están usando para hacerla. Esta librería nos provee una clase que podemos usar para aplicar en el `setOnTouchListener()` de cualquier objeto visual de nuestra app.

Sensores utilizados

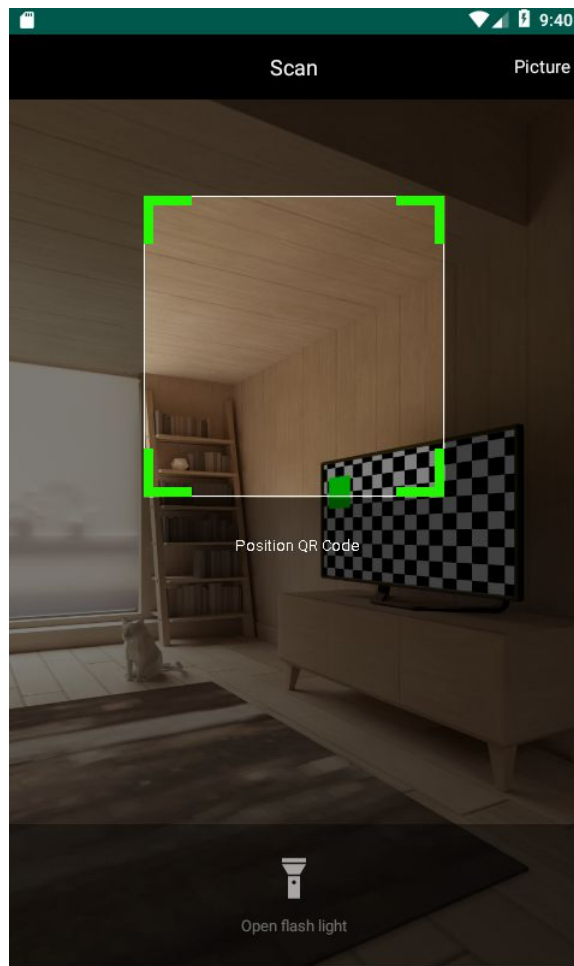
Cámara-Lector de Qr

Para la lectura de Qr, hemos usado la librería QRCodeScanner la cual implementamos de la siguiente manera.

Lo primero que realizamos es pedir permisos, si la API es superior a la **LOLLIPOP_MR1** tendremos que pedir permisos en caso de que no estuvieran dados, si es menor o igual, nos vale con que los permisos se incluyan en el Manifest.

```
if (Build.VERSION.SDK_INT > Build.VERSION_CODES.LOLLIPOP_MR1) { // Marshmallow+
    if (ContextCompat.checkSelfPermission(this, Manifest.permission.CAMERA) !=
        PackageManager.PERMISSION_GRANTED) {
        // Should we show an explanation?
        if (ActivityCompat.shouldShowRequestPermissionRationale(this, Manifest.permission.CAMERA)) {
            // Show an explanation to the user *asynchronously* -- don't block
            // this thread waiting for the user's response! After the user
            // sees the explanation, try again to request the permission.
        } else {
            // No se necesita dar una explicación al usuario, sólo pedimos el permiso.
            ActivityCompat.requestPermissions(this, new String[]{Manifest.permission.CAMERA},
                MY_PERMISSIONS_REQUEST_CAMARA);
            // MY_PERMISSIONS_REQUEST_CAMARA es una constante definida en la app. El método callback
            // obtiene el resultado de la petición.
        }
    } else { // have permissions
        Intent i = new Intent(MainActivity.this, QrCodeActivity.class);
        startActivityForResult(i, REQUEST_CODE_QR_SCAN);
    }
} else { // Pre-Marshmallow
    Intent i = new Intent(MainActivity.this, QrCodeActivity.class);
    startActivityForResult(i, REQUEST_CODE_QR_SCAN);
}
```

Si tenemos los permisos o si al volver de pedirlos, los tenemos lanzamos un intent que nos manda a una activity con la cámara del móvil para leer el Qr.



Cuando esta activity detecta el código QR, nos devuelve a la activity anterior y nos manda como extra la información leída.

Nuestros QR tienen el formato: tipo:id. Donde tipo puede ser obra, sala, colección y el id es su respectivo id.

```
if(resultCode != Activity.RESULT_OK)
{
    Log.d(LOGTAG, "COULD NOT GET A GOOD RESULT.");
    if(data==null)
        return;
    //Getting the passed result
    String result = data.getStringExtra("com.blikoon.qrcodescanner.error_decoding_image");
    if(result!=null)
    {
        AlertDialog alertDialog = new AlertDialog.Builder(MainActivity.this).create();
        alertDialog.setTitle("Scan Error");
        alertDialog.setMessage("QR Code could not be scanned");
        alertDialog.setButton(AlertDialog.BUTTON_NEUTRAL, "OK",
            new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int which) {
                    dialog.dismiss();
                }
            });
        alertDialog.show();
    }
    return;
}
```

```

}
if(requestCode == REQUEST_CODE_QR_SCAN)
{
    if(data==null)
        return;
    //Getting the passed result
    String result = data.getStringExtra("com.blikoon.qrcodescanner.got_qr_scan_result");
    manejoEtiqueta(result);
}

```

Si el resultado es correcto, el requestCode es igual al **REQUEST_CODE_QR_SCAN** y los datos son distintos de null, ejecutamos el metodo manejoEtiqueta(String text). La cual se encarga de detectar qué es lo que hay que hacer con esa información y ejecutar la activity necesaria. Este metodo tambien se usara a la hora de leer etiquetas NFC que describiremos más adelante.

```

protected void manejoEtiqueta(String text){
    String[] registros = text.split(":");
    String tipo = registros[0];
    int id = Integer.parseInt(registros[1]);
    if(tipo.toLowerCase().equals(TIPO_OBRA)){
        ms.getObraId(id);
        Intent intent = new Intent(getApplicationContext(), Informacion_sobre_obra.class);
        intent.putExtra("museo",ms);
        startActivity(intent);
    }else if ( tipo.toLowerCase().equals(TIPO_SALA)){
        Intent intent = new Intent(getApplicationContext(), VisualizacionObras.class);
        intent.putExtra("museo",ms);
        intent.putExtra("id_sala",id);
        startActivity(intent);
    }else if ( tipo.toLowerCase().equals(TIPO_COLECCION)){
        Intent intent = new Intent(getApplicationContext(), VisualizacionObras.class);
        intent.putExtra("museo",ms);
        intent.putExtra("id_coleccion",id);
        startActivity(intent);
    }
}
}

```

NFC

Para usar el lector NFC de nuestro dispositivo, lo primero que tenemos que hacer es añadir los siguientes permisos de usuario al Manifest.

```

<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.NFC" />

```

El primero impide que el dispositivo entre en modo suspensión, el segundo permite el uso del NFC.

Lo siguiente que realizamos en el Manifest es añadir:

```
<intent-filter>
  <action android:name="android.nfc.action.TECH_DISCOVERED" />

  <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
```

```
<meta-data
  android:name="android.nfc.action.TECH_DISCOVERED"
  android:resource="@xml/nfc_tech_list" />
```

El intent-filter nos permite que cuando el dispositivo descubra una etiqueta NFC, nos de la posibilidad de manejar esos datos con nuestra aplicación. El manejo de ese intent se realiza en el `onNewIntent(Intent intent)`.

`@Override`

```
protected void onNewIntent(Intent intent) {
  super.onNewIntent(intent);

  if (NfcAdapter.ACTION_TECH_DISCOVERED.equals(intent.getAction())) {
    Parcelable[] rawMessages = intent.getParcelableArrayExtra(NfcAdapter.EXTRA_NDEF_MESSAGES);
    if (rawMessages != null) {
      NdefMessage[] rawMsgs = new NdefMessage[rawMessages.length];
      for (int i = 0; i < rawMessages.length; i++) {
        rawMsgs[i] = (NdefMessage) rawMessages[i];
      }
      // only one message sent during the beam
      NdefMessage msg = (NdefMessage) rawMsgs[0];
      String s = new String(msg.getRecords()[0].getPayload());
      //Elimino los 3 primeros caracteres que aparecen ya que no son validos,
      // no entiendo muy bien el porque ocurre
      manejoEtiqueta(s.substring(3));
    }
  }
}
```

Si la acción que lanza el intent es del tipo `NfcAdapter.ACTION_TECH_DISCOVERED`, lo manejamos obteniendo los datos del `NdefMessage` y mandándolo a la función `manejoEtiqueta()`, descrita en el apartado anterior.

Multitouch y gestos

Para la detección del multitouch y los gestos usamos la librería `SimpleFingerGestures`. Esta librería nos proporciona una clase llamada `SimpleFingerGestures`, a la cual tenemos que implementar los métodos a realizar cuando se reconozca alguno de los gestos en la pantalla. Para usar esta clase lo primero que tenemos que hacer es crear un objeto de la clase y decir que va a ser el que consume el evento.

```
SimpleFingerGestures mySfg = new SimpleFingerGestures();
mySfg.setConsumeTouchEvents(true);
```

Ahora tenemos que implementar los métodos abstractos que nos da la clase para que hacer según el gesto que se reconoce.

```

mySfg.setOnFingerGestureListener(new SimpleFingerGestures.OnFingerGestureListener() {
    @Override
    public boolean onSwipeUp(int fingers, long gestureDuration, double gestureDistance) {

        if(fingers == 3){
            FloatingActionButton speak = findViewById(R.id.id_listener);
            speak.setEnabled(true);
            speak.setBackgroundTintList(ColorStateList.valueOf(Color.RED));
            speak.setRippleColor(ColorStateList.valueOf(Color.GREEN));
            speak.show();
            iniciarTTS();

        }
        return false;
    }

    @Override
    public boolean onSwipeDown(int fingers, long gestureDuration, double gestureDistance) {
        FloatingActionButton speak = findViewById(R.id.id_listener);
        speak.setEnabled(false);
        if (fingers == 3 && speak.isEnabled()) {
            shutdown();
            speak.setEnabled(false);
            speak.hide();
        }
        return false;
    }

    @Override
    public boolean onSwipeLeft(int fingers, long gestureDuration, double gestureDistance) {
        if(fingers == 1){
            ms.getObraSiguiente();
            loadObraActual();
        }
        return false;
    }

    @Override
    public boolean onSwipeRight(int fingers, long gestureDuration, double gestureDistance) {
        if(fingers == 1){
            ms.getObraAnterior();
            loadObraActual();
        }
        return false;
    }
});

```

Hemos usado los metodos para deslizar hacia arriba, abajo y los lados. El de hacia arriba y abajo lo hemos usado para quitar el boton del asistente virtual y pararlo en caso de que este en uso. El de hacia los lados lo hemos usado para pasar entre obras.

Acelerómetro

Para el acelerómetro usaremos las variables:

```
private SensorManager senSensorManager;  
private Sensor senAccelerometer;
```

En ellas se guardara el sensor que estamos usando:

```
senSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);  
senAccelerometer = senSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);  
senSensorManager.registerListener(this, senAccelerometer, SensorManager.SENSOR_DELAY_NORMAL);
```

Cuando haya un cambio de estado en alguno de los sensores se ejecutara el siguiente método;

```
@Override  
public void onSensorChanged(SensorEvent sensorEvent) {  
  
    Sensor mySensor = sensorEvent.sensor;  
  
    if (mySensor.getType() == Sensor.TYPE_ACCELEROMETER) {  
        float x = sensorEvent.values[0];  
        float y = sensorEvent.values[1];  
        float z = sensorEvent.values[2];  
        long curTime = System.currentTimeMillis();  
  
        if ((curTime - lastUpdate) > 200) {  
            long diffTime = (curTime - lastUpdate);  
            lastUpdate = curTime;  
            float speed = Math.abs(x + y + z - last_x - last_y - last_z) / diffTime * 10000;  
  
            if (speed > SHAKE_THRESHOLD) {  
                cambiarEstadoBoton();  
            }  
  
            last_x = x;  
            last_y = y;  
            last_z = z;  
        }  
    }  
}
```

Este método detecta los movimientos que realiza el dispositivo y calcula la velocidad con el que se a realizado, si la velocidad es mayor a la establecida en la variable

SHAKE_THRESHOLD, entonces ejecutaremos el método `cambiarEstadoBoton()`, la cual lo que realiza es callar al TTS, en caso de que esté hablando. Eso lo hacemos así para mejorar la usabilidad del cliente.

Proximidad

La implementación del sensor de proximidad se realiza de forma muy parecida al acelerómetro. Declaramos las variables:

```
private Sensor mProximity;
```

```
private static final int SENSOR_SENSITIVITY = 4;
```

Donde SENSOR_SENSITIVITY, será una variable que indica la distancia en la que se pasa de un estado a otro. La función que controla cuando cambia un sensor su estado al igual que con el acelerómetro es **public void** onSensorChanged(SensorEvent sensorEvent). A la cual le hemos añadido a parte del código del acelerómetro lo siguiente:

```
if (sensorEvent.sensor.getType() == Sensor.TYPE_PROXIMITY) {  
    AudioManager audioManager = (AudioManager) getSystemService(AUDIO_SERVICE);  
  
    if (sensorEvent.values[0] >= -SENSOR_SENSITIVITY && sensorEvent.values[0] <= SENSOR_SENSITIVITY)  
    {  
        cambiarEstadoBoton();  
    }  
}
```

Es decir cuando el sensor esté a una determinada distancia, ejecuta la función que llama al TTS.

Implementación de Dialog Flow

Para la implementación de Dialog Flow en Android hemos usado el material proporcionado por la profesora Dra. Zoraida Callejas Carrión, más concretamente el proyecto ChatBot. El proyecto proporcionaba una forma fácil de conexión entre Android y Dialog Flow a partir de la API que nos proporciona Dialog Flow.

Se ha utilizado el propio inline editor proporcionado por Dialog Flow para implementar las funciones de los intents que necesitan información sobre una obra. La información sobre las obras, salas y colecciones están almacenadas en dos vectores. Con esto nos evitamos tener intents específicos para la información de cada obra sala o colección.

Dialog Flow

Almacenamiento de la información.

Por problemas de conexión con la base de datos y tiempos altos al cargar los datos la información ha sido guardada en dos vectores que se explicarán a continuación.

vector de salas

```
var arrSalas = [];  
arrSalas[0] = [];  
arrSalas[0][0] = 'sala magna';  
arrSalas[0][1] = 'Rodriguez Acosta';  
arrSalas[0][2] = 0;  
arrSalas[0][3] = 'pertenece a José María Rodríguez-Acosta González de la Cámara que nació';  
arrSalas[0][4] = false;
```

El primer campo del vector interior de arrSalas representa el nombre de la sala, el segundo es el nombre de la colección, el tercero es el número de colección, el cuarto la descripción de la colección y el quinto campo es el booleano que usaremos para saber si una sala fue visitada o no.

Vector de cuadros

```
43 var arrCuadros = [];  
44 //var cuadro = [];  
45 arrCuadros[0] = [];  
46 arrCuadros[0][0] = 'virgen entronizada';  
47 arrCuadros[0][1] = 0;  
48 arrCuadros[0][2] = 'Pequeña escultura realizada en marfil, de virgen entronizada, senta';  
49 arrCuadros[0][3] = 'antonio lopez';  
50
```

El primer campo de vector interior de arrCuadros representa el nombre de la obra, el segundo es el id de la sala a la que pertenece, el tercero es la descripción de la obra y el cuarto es el nombre del autor.

Entities

Los entities han sido utilizados con dos propósitos:

1. Sinónimos de palabras muy utilizadas en nuestra aplicación.

En este caso se han utilizado para palabras que el usuario puede utilizar y que cada usuario puede tener preferencias por usar unas u otras.

☒ Define synonyms ☐ Allow automated expansion

autor	autor, creador, artista
Click here to edit entry	

TiposObra

SAVE

☒ Define synonyms ⓘ
 ☐ Allow automated expansion

obra	obra
cuadro	cuadro, retrato
obras	obras
Click here to edit entry	

2. Información específica del museo

En este caso se usan los entities para reconocer las salas colecciones y obras del propio museo dentro de los intents.

autores

SAVE

☒ Define synonyms ⓘ
 ☐ Allow automated expansion

antonio lopez	antonio lopez
juan gris	juan gris
miguel de cervantes	miguel de cervantes
jose de ribera	jose de ribera
Click here to edit entry	

colecciones

SAVE



☒ Define synonyms ⓘ
 ☐ Allow automated expansion

Rodriguez Acosta	Rodriguez Acosta, rodriguez acosta, Rodriguez Acosta, Rodriguez acosta
Gomez Moreno	Gomez Moreno, gomez moreno, Gomez moreno, gomez Moreno
Arte Moderno	arte moderno
Click here to edit entry	

obras_arte

SAVE

☒ Define synonyms ⓘ
☐ Allow automated expansion

autorretrato	autorretrato, autoretrato	 
andaluzas	andaluzas	
abril	abril	
virgen entronizada	virgen entronizada	
caida de faeton	caída de faetón, caída de faeton, caída de faeton, caída de faetón	
sagrada familia	sagrada familia	
angel	ángel, angel	
san juan bautista	san juan bautista	
el quijote	el quijote	
parnasos	parnasos	
capitel nazari	capitel nazari, capitel nazari	
santa margarita	santa margarita	

Click here to edit entry

salas

SAVE

☒ Define synonyms ⓘ
☐ Allow automated expansion

Click here to edit entry

Intents

Ha la hora de realizar los intents se han tenido en cuenta varios bloques principales de los que costaría nuestra aplicación.

Guia

Se consta de una guía que se encargará de llevar un registro de las salas ya visitadas y dar información específica de la sala actual. Esta guía no será obligatoria de iniciar si el usuario no lo cree oportuno. Para esto se necesita una variable global para el agente que contenga la sala actual. Esta variable será llamada salaActual;

```
18 var ultimaObra, salaActual;
```

En caso de ser utilizada se iniciará con el intent IniciarGuia que cambiará el contexto de salida a peticiónSala.

IniciarGuia

SAVE

Contexts ?

^

Add input context

5 peticionSala Add output context

Events ?

^

Training phrases ?

Search training phrases

Q

^

Add user expression

empezar guia

guiame

iniciar guia

Una vez invocado el intent el asistente nos preguntará con el apartado Responses porqué sala empezar

Responses ?

^

DEFAULT GOOGLE ASSISTANT +

Text response

?

🗑

1 ¿Por que sala empezamos?

2 ¿Dime una sala por la que empezar?

3 ¿Por que sala quiere empezar?

4 Enter a text response variant

ADD RESPONSES

Llegados a este punto tendremos dos posibles elecciones que contendrán el contexto de entrada peticiónSala:

1. No dar una sala específica:

En este caso asignaremos la primera sala como la sala de inicio. Será tratado por el intent NoImportaSala.

NolImportaSala

SAVE

Contexts ⓘ

peticiónSala ⓘ Add input context

5 guíaIniciada ⓘ Add output context

Events ⓘ

Training phrases ⓘ

Search training phrases 🔍

” Add user expression

” no importa

” cualquiera

” la que tu quieras

” tu eliges la sala

” No importa la sala

Se añade al map función que devolverá la respuesta del agente a el inline editor y se implementa para que ponga la sala actual como la primera sala del museo.

```
556 intentMap.set('NoImportaSala', salaInicial);

391 function salaInicial (agent) {
392   salaActual = 0;
393   arrSalas[0][4] = true;
394   agent.add('Empezaremos por la ' + arrSalas[salaActual][0] + ' que contiene la coleccion ' + arrSalas[salaActual][1] + '. Dirijase a ella' );
395 }
...
```

2. Dar una sala específica.

En este caso si se especifica la sala por la que el usuario quiere empezar mediante el intent PeticionSala.

PeticionSala

SAVE

Contexts ⓘ

peticionSala ⓘ Add input context

5 guiaIniciada ⓘ Add output context

Events ⓘ

Training phrases ⓘ

Search training phrases 🔍

” Add user expression

” quiero empezar por la sala

” por la sala

” empieza por sala

” sala

Training phrases ⓘ

Search training phrases 🔍

” Add user expression

” quiero empezar por la sala

” por la sala

” empieza por sala

” sala

Action and parameters ⓘ

Enter action name

REQUIRED ⓘ	PARAMETER NAME ⓘ	ENTITY ⓘ	VALUE	IS LIST ⓘ
<input type="checkbox"/>	salas	@salas	\$salas	<input type="checkbox"/>
<input type="checkbox"/>	Enter name	Enter entity	Enter value	<input type="checkbox"/>

+ New parameter

Se añade al map función que devolverá la respuesta del agente utilizando la sala dada por el usuario.

```
557 intentMap.set('PeticiónSala', cambiarSala);

397 ~ function cambiarSala (agent) {
398 ~     var i = 0;
399 ~     while(arrSalas[i][0] != agent.parameters['salas']){
400 ~         i++;
401 ~     }
402 ~     salaActual = i;
403 ~     /*for(var i = 0 ; i < arrSalas.length; i++){
404 ~         if(arrSalas[i][0] == agent.parameters['salas'] ){
405 ~             salaActual = i;
406 ~         }
407 ~     }*/
408 ~     if(arrSalas[salaActual][4]){
409 ~         agent.add( 'Esta ' + arrSalas[salaActual][0] + ' contiene la colección ' + arrSalas[salaActual][1] + ' que ya la tienes marcada como vis:
410 ~     }else{
411 ~         agent.add( 'Esta ' + arrSalas[salaActual][0] + ' contiene la colección ' + arrSalas[salaActual][1]);
412 ~     }
413 ~     arrSalas[salaActual][4] = true;
414 ~
415 ~ }
```

También se da la opción de que un usuario introduzca una sala como sala actual sin la necesidad de hacerlo mediante la opción de iniciar guía con el intent CambiarSalaActual.

CambiarSalaActual

SAVE

Training phrases ?

Search training phrases

” Add user expression

” avanzamos a la sala

” pasamos a la sala

” cambia a la sala

” cambiamos a la sala

Action and parameters

Enter action name

REQUIRED ?	PARAMETER NAME ?	ENTITY ?	VALUE	IS LIST ?
<input type="checkbox"/>	salas	@salas	\$salas	<input type="checkbox"/>
<input type="checkbox"/>	Enter name	Enter entity	Enter value	<input type="checkbox"/>

Esto se realizará mediante la función cambiarSala. Siempre que se cambien a una sala se tendrá en cuenta si esta sala ha sido ya visitada para informar al usuario y se marcará como visitada.

```
558 intentMap.set('CambiarSalaActual', cambiarSala);

397 function cambiarSala (agent) {
398   var i = 0;
399   while(arrSalas[i][0] != agent.parameters['salas']){
400     i++;
401   }
402   salaActual = i;
403   /*for(var i = 0 ; i < arrSalas.length; i++){
404     if(arrSalas[i][0] == agent.parameters['salas'] ){
405       salaActual = i;
406     }
407   }*/
408   if(arrSalas[salaActual][4]){
409     agent.add( 'Esta ' + arrSalas[salaActual][0] + ' contiene la colección ' + arrSalas[salaActual][1] + ' que ya la tienes marcada como vis:
410   }else{
411     agent.add( 'Esta ' + arrSalas[salaActual][0] + ' contiene la colección ' + arrSalas[salaActual][1]);
412   }
413   arrSalas[salaActual][4] = true;
414 }
415 }
```

Una vez iniciada la guía o introducida una sala por el usuario, tendremos distintos intents para obtener información sobre la sala actual como puede ser, listar las obras de esta sala, información de esta sala y preguntar cual es la sala actual.

```
559 intentMap.set('InformacionSalaActual', informacionSalaActual);
560 intentMap.set('ObrasSalaActual', obrasSalaActual);
561 intentMap.set('SalaActual', salaUbicacionActual);
```

Siempre se realizará la comprobación de si la sala actual está definida por si el usuario no la inicio de ninguna de las formas establecidas anteriormente.

```
441 if(typeof salaActual !== "undefined"){
```

También se tiene la opción de cambiar de sala llamando al intent SiguienteSala que pasará a la sala siguiente de la actual.

• SiguienteSala

SAVE

Contexts ?

Events ?

Training phrases ?

Search training phrases

Add user expression

sala siguiente

siguiente sala

pasar a la siguiente sala

Se realizará mediante la función siguienteSala que cogerá la posición siguiente a la sala actual del array de salas. Si no se a dado una sala actual aun se pondrá la sala actual como la primera sala.

```
564 intentMap.set('SiguienteSala', siguienteSala);

499 ~ function siguienteSala (agent) {
500 ~   if(typeof salaActual !== "undefined"){
501 ~     salaActual = (salaActual + 1) % arrSalas.length;
502 ~   }
503 ~   }else{
504 ~     salaActual = 0;
505 ~   }
506 ~
507 ~   if(arrSalas[salaActual][4]){
508 ~     agent.add('la siguiente sala es: ' + arrSalas[salaActual][0] + 'que contiene la coleccion ' + arrSalas[salaActual][1] + ' que ya la tier
509 ~   }else{
510 ~     agent.add('la siguiente sala es: ' + arrSalas[salaActual][0] + 'que contiene la coleccion ' + arrSalas[salaActual][1]);
511 ~   }
512 ~   arrSalas[salaActual][4] = true;
513 ~ }
```

Como ultima opcion de nuestra guia damos al usuario la posibilidad de listar las salas que no ha visitado aún mediante el intent SalasNoVisitadas.

- SalasNoVisitadas

SAVE

Contexts ?

Events ?

Training phrases ?

Search training phrases

” Add user expression

” que salas no he visitado aun

” salas por visitar

” salas sin visitar

” salas no visitadas

” dime las salas que no he visitado

” que salas me quedan por visitar

Esta funcionalidad simplemente recorre el array salas de forma que guarda las que no estén marcadas como visitadas.

```
565 intentMap.set('SalasNoVisitadas', salasSinVisitar);
```

Información sobre Salas

En el caso de las salas solo tenemos tres intents que muestran información sobre estas.

Listar los cuadros de una sala.

Se realiza mediante el intent ListarCuadrosSalaNom y el agente da como salida los cuadros de la sala pasada por el usuario.

ListarCuadrosSalaNom

SAVE

Try it

Contexts ?

Events ?

Training phrases ?

Search training phrases

» Add user expression

» dime los cuadros de la sala

» que cuadros tiene la sala

» cuadros de sala

» que cuadros hay en la sala

Action and parameters

Enter action name

REQUIRED ?	PARAMETER NAME ?	ENTITY ?	VALUE	IS LIST ?
<input type="checkbox"/>	salas	@salas	\$salas	<input type="checkbox"/>
<input type="checkbox"/>	TiposObra	@TiposObra	\$TiposObra	<input type="checkbox"/>

```
544 intentMap.set('ListarCuadrosSalaNom', obtenerCuadrosSala);
```

Listar las salas con las colección que contiene.

Simplemente lista la salsa que tenemos con la colección que corresponde a cada sala. En este caso no se necesita ningún entity para realizar esta acción. La función recorrerá el vector de Salas quedándose con el nombre de la sala y de la colección y esta será la salida del agente.

• ListarColeccionesySalas

SAVE

Training phrases ⓘ

Search training phrases



” Add user expression

” listar salas y colecciones

” listar colecciones

” listar salas

” dime las salas que tenemos

” que salas tenemos

” que salas hay

” salas del museo

” salas

” colecciones del museo

” que colecciones hay

1 OF 2



```
551 intentMap.set('ListarColeccionesySalas', listarSalas);
```

Descripción de una sala

En este caso el usuario dirá el nombre de la sala y se tendrá que recuperar de los parámetros del agente para poder quedarnos con la descripción de la sala del vector con ese nombre.

• DescripcionSala

SAVE

Training phrases ⓘ

Search training phrases



” Add user expression

” de que es la sala

” que hay en la sala

” descripción de la sala

Action and parameters



Enter action name

REQUIRED ⓘ	PARAMETER NAME ⓘ	ENTITY ⓘ	VALUE	IS LIST ⓘ
<input type="checkbox"/>	salas	@salas	\$salas	<input type="checkbox"/>
<input type="checkbox"/>	Enter name	Enter entity	Enter value	<input type="checkbox"/>

+ New parameter

```
562 intentMap.set('DescripcionSala', descripcionSala);
```

Información sobre colecciones

Descripción de una colección

El usuario pasa mediante el intent `DescripcionColeccionNom` el nombre de la colección de la que el usuario quiere saber la descripción y mediante la función `descripcion` se recoge de los parámetros del agente el parámetro `coleccion` y se compara con las colecciones de las salas para quedarnos con la descripción de la sala en la que se encuentre la colección

• DescripcionColeccionNom

[SAVE](#)

Training phrases ⓘ



” Add user expression

” descripción de la colección Rodríguez Acosta

” dime la descripción de la colección Rodríguez Acosta

” colección Rodríguez Acosta

” que hay en la colección Rodríguez Acosta

” de que es la colección Rodríguez Acosta

Action and parameters

REQUIRED ⓘ	PARAMETER NAME ⓘ	ENTITY ⓘ	VALUE	IS LIST ⓘ
<input type="checkbox"/>	coleccion	@coleccion	\$coleccion	<input type="checkbox"/>
<input type="checkbox"/>	Enter name	Enter entity	Enter value	<input type="checkbox"/>

[+ New parameter](#)

```
541 intentMap.set('DescripcionColeccionNom', descripcion);
```

Obtene obras de una colección

En este caso igual que el anterior se pasa el nombre de la colección mediante el intent y se compara con el campo `coleccion` del array `salas`. De esta forma nos quedaremos con las salas en las que está la colección para luego compararlos con el campo `sala` del array `cuadros`.

Training phrases

Search training phrases

Add user expression

descripción de la colección Rodríguez Acosta

dime la descripción de la colección Rodríguez Acosta

colección Rodríguez Acosta

que hay en la colección Rodríguez Acosta

de que es la colección Rodríguez Acosta

Action and parameters

Enter action name

REQUIRED	PARAMETER NAME	ENTITY	VALUE	IS LIST
<input type="checkbox"/>	colecciones	@colecciones	\$colecciones	<input type="checkbox"/>

```

543 intentMap.set('ListarCuadrosColeccionNom', obtenerCuadros);

169 function obtenerCuadros (agent) {
170     var cuadros = '';
171     var coleccion = agent.parameters['colecciones'];
172     var numSala, encontrado = false;
173
174     for(var j = 0; j < arrSalas.length && !encontrado; j++){
175         if (coleccion === arrSalas[j][0] || coleccion === arrSalas[j][1] || coleccion === arrSalas[j][2] ){
176             numSala = arrSalas[j][2];
177             encontrado = true;
178         }
179     }
180     if(encontrado){
181         for(var i = 0; i < arrCuadros.length; i++){
182             if(numSala === arrCuadros[i][1] ){
183                 if(i == (arrCuadros.length - 1)){
184                     cuadros += ' y ' + arrCuadros[i][0] + ',';
185                 }else{
186                     cuadros += arrCuadros[i][0] + ', ';
187                 }
188             }
189         }
190         agent.add('Las obras son los siguientes: ' + cuadros);
191     }else{
192         agent.add('No existe esa coleccion');
193     }
194 }
195

```

Información de obras

Descripción, sala, colección y descripción obra específica

En este caso se le pasará a el nombre de la obra en el intent y se recupera de los parámetros del agente para compararlo con el vector de cuadros. Se añade un contexto de salida (obraAnterior) que nos servirá para poder obtener información de este mismo cuadro

sin dar el título obligatoriamente. En este caso se puede ver en el ejemplo del intent de descripción de una obra.

• DescripcionObraNom

SAVE

Contexts ?

Add input context

5 obraAnterior Add output context

Events ?

Training phrases ?

Search training phrases

” Add user expression

” dime la descripción de la obra reloj

” descripción de la obra reloj

” dime la descripción de reloj

” descripción de reloj

” reloj descripción

En la función se recogerá de los parámetros del agente el nombre del cuadro para compararlo con el array de cuadros y quedarnos con la información necesaria.

```
546 intentMap.set('AutorObraNom', autorObra);
548 intentMap.set('DescripcionObraNom', descripcionObra);
549 intentMap.set('SalaObraNom', salaObra);
550 intentMap.set('ColeccionObraNom', coleccionObra);
```

Basándonos en el ejemplo del autor vamos a ver cómo se almacena el nombre la obra en una variable global llamada ultimaObra que se utilizara para poder ver los detalles del cuadro consultado sin decir el nombre específico de ese. Se cogerá el nombre de la última obra consultada mediante uno de estos intents.

```
18 var ultimaObra, salaActual;

243 function autorObra (agent) {
244     var encontrado = false;
245     var nombre = agent.parameters['obrasArte'];
246     //agent.add('hola' + nombre);
247     for(var j = 0; j < arrCuadros.length && !encontrado;j++){
248         if ( nombre === arrCuadros[j][0]){
249             agent.add( 'El autor es ' + arrCuadros[j][3] + '.');
250             encontrado = true;
251             ultimaObra = nombre;
252         }
253     }
254     if(!encontrado){
255         agent.add('El cuadro ' + nombre + ' no pertenece a ninguna de nuestras colecciones');
256     }
257 }
258 }
```

Descripción, sala, colección y descripción obra anterior

En este caso se utiliza la variable ultimaObra para obtener la información de la obra con ese título. El contexto de entrada del intent es obraAnterior y se coloca también obraAnterior como contexto de salida por si se quiere volver a preguntar sobre información de ese mismo cuadro sin tener que especificar el nombre.

- descripcionObraAnterior

SAVE

Contexts ?

obraAnterior  Add input context

5 obraAnterior  Add output context 

Events ?

Training phrases ?

Search training phrases



” Add user expression

” descripcion de la obra

” descripcion

Las funciones en este caso comparan la variable ultimaObra para seleccionar la información pedida por el usuario.

```
547 intentMap.set('AutorObraAnterior', autorAnterior);

552 intentMap.set('descripcionObraAnterior', descripcionAnterior);
553 intentMap.set('ColeccionObraAnterior', coleccionAnterior);
554 intentMap.set('SalaObraAnterior', salaAnterior);

325 function descripcionAnterior (agent) {
326     var encontrado = false;
327     //var nombre = agent.parameters['obrasArte'];
328     //agent.add('hola' + nombre);
329     for(var j = 0; j < arrCuadros.length && !encontrado;j++){
330         if ( ultimaObra === arrCuadros[j][0]){
331             agent.add(arrCuadros[j][2] + '.');
332             encontrado = true;
333         }
334     }
335     if(!encontrado){
336         agent.add('El cuadro ' + ultimaObra + ' no pertenece a ninguna de nuestras colecciones');
337     }
338 }
339 }
```

Nombre de la obra actual.

El usuario puede llegar a preguntarse en llegado el momento sobre que obra esta pidiendo informacion al pedirla sin especificar el nombre. Para eso añadiremos un intent con el contexto de salida y de entrada obraAnterior.

ObraActual

SAVE

Contexts

obraAnterior

Add input context

5 obraAnterior

Add output context

Training phrases

Search training phrases

Add user expression

que obra es

de que obra me hablas

obra actual

Action and parameters

Enter action name

REQUIRED	PARAMETER NAME	ENTITY	VALUE	IS LIST
<input type="checkbox"/>	TiposObra	@TiposObra	\$TiposObra	<input type="checkbox"/>

En este caso la función simplemente muestra el nombre almacenado en la variable ultimaObra.

```
555 intentMap.set('ObraActual', obraActual);  
  
386 function obraActual (agent) {  
387   agent.add('La obra es ' + ultimaObra);  
388 }
```

Listar obras de un autor

Se da el nombre de un autor para buscar las obras realizadas por este. En este intents como se ve en la imagen es necesarios usar tres tipos de entities. Autores que contienen los autores de nuestro museo, autor que contiene sinónimos posibles para autor y tipoObra que contiene los sinónimos de obras.

• ListarObrasAutor

SAVE

Training phrases ?

Search training phrases



” Add user expression

” dime las obras realizas por miguel de cervantes

” listar obras de miguel de cervantes

” obras de miguel de cervantes

” obras del autor autor

Action and parameters



Enter action name

REQUIRED ?	PARAMETER NAME ?	ENTITY ?	VALUE	IS LIST ?
<input type="checkbox"/>	autores	@autores	\$autores	<input type="checkbox"/>
<input type="checkbox"/>	autor	@autor	\$autor	<input type="checkbox"/>
<input type="checkbox"/>	TiposObra	@TiposObra	\$TiposObra	<input type="checkbox"/>
<input type="checkbox"/>	Enter name	Enter entity	Enter value	<input type="checkbox"/>

La función recoge de los parámetros del agente el parámetro autores que comparara con el campo autores del vector de cuadros.

```
563 intentMap.set('ListarObrasAutor', listarObrasAutor);
```

Información completa obra

Muestra toda la información de una obra en un solo mensaje del agente. Al finalizar de mostrar la información realiza una pregunta para saber si se quiere saber la información completa de otra obra. tiene los contextos de salida obraAnterior, para poder realizar preguntas sin especificar el título después de esta pregunta, y información de obra que será la que se encargue de procesar la respuesta del usuarios a la pregunta realizada anteriormente por el agente.

• InformacionCompletaObraNom

SAVE

Contexts ?



Add input context

5 informacionObra 5 obraAnterior Add output context

” información completa de la obra reloj
” información completa obra
” información completa de obra
” reloj información
” reloj datos
” datos reloj
” reloj
” información reloj
” información sobre reloj

Action and parameters

REQUIRED ?	PARAMETER NAME ?	ENTITY ?	VALUE	IS LIST ?
<input type="checkbox"/>	obrasArte	@obras_arte	\$obrasArte	<input type="checkbox"/>
<input type="checkbox"/>	obras_arte	@obras_arte	\$obras_arte	<input type="checkbox"/>
<input type="checkbox"/>	TiposObra	@TiposObra	\$TiposObra	<input type="checkbox"/>
<input type="checkbox"/>	Enter name	Enter entity	Enter value	<input type="checkbox"/>

La función que se encarga de esto es informaciónCompleta que se encarga de añadir al final de la información del cuadro la pregunta para saber si se quiere la información completa de otra obra

```

225 function informacionCompleta (agent) {
226     var encontrado = false;
227     var nombre = agent.parameters['obrasArte'];
228     //agent.add('hola' + nombre);
229     for(var j = 0; j < arrCuadros.length && !encontrado;j++){
230         if ( nombre === arrCuadros[j][0]){
231             agent.add( 'El autor es ' + arrCuadros[j][3] + '. La descripción de la obra es: ' + arrCuadros[j][2]
232             + '. Se expone en la colección ' + arrSalas[arrCuadros[j][1]][1] + ' situada en ' + arrSalas[arrCuadros[j][1]][0]
233             + '. ¿Quiere informacion de otra obra? ');
234             encontrado = true;
235             ultimaObra = nombre;
236         }
237     }
238     if(!encontrado){
239         agent.add('El cuadro ' + nombre + ' no pertenece a ninguna de nuestras colecciones. ¿Quiere informacion de otra obra?');
240     }
241 }
242
243

```

Después de realizar la pregunta el usuario tendrás dos opciones:

1. No querer más información de otra obra.

En este caso se activaría el intent NoInformacionObra que tendrá como contexto de entrada informacionObra y como contexto de salida obraAnterior para poder preguntar información de la obra sin dar el nombre. En este caso este intent no tiene una función sino que responde a través del propio intent

• NoInformacionObra

SAVE

Contexts ⓘ

informacionObra ⓘ Add input context

5 obraAnterior ⓘ Add output context

Events ⓘ

Training phrases ⓘ

Search training phrases 🔍

” Add user expression

” claro que no

” en otra ocasión

” no

Responses ⓘ

DEFAULT GOOGLE ASSISTANT +

Text response ⓘ 🗑

1 Perfecto. En otra ocasión.

2 Enter a text response variant.

ADD RESPONSES

☐ Set this intent as end of conversation ⓘ

2. Si quieres información completa sobre otra obra

En este caso también se tendrá como contexto de entrada informacionObra y de salida obraAnerior como en el caso anterior.

• SiInformacionCuadro

SAVE

Contexts ?

informacionObra ⓘ Add input context

5 obraAnerior ⓘ Add output context

Events ?

Training phrases ?

Search training phrases 🔍

” Add user expression

” claro

” si

Responses ?

DEFAULT GOOGLE ASSISTANT +

Text response ⓘ 🗑

1 Dime el nombre de la obra

2 Enter a text response variant

ADD RESPONSES

Al contestar a este intent con el nombre la obra saltara el intent información completa que nos mostrará la información de esa obra al aceptar cadenas del siguiente estilo.

” reloj información

” reloj datos

” datos reloj

” reloj

” información reloj

” información sobre reloj 🗑

Referencias

1. [Picasso](http://square.github.io/picasso/) <http://square.github.io/picasso/>
2. [QRCodeScanner](https://github.com/blikoon/QRCodeScanner) <https://github.com/blikoon/QRCodeScanner>
3. [Firebase Database](https://firebase.google.com/docs/android/setup?hl=es-419) <https://firebase.google.com/docs/android/setup?hl=es-419>
4. [SimpleFingerGestures](https://github.com/championswimmer/SimpleFingerGestures_Android_Library)
https://github.com/championswimmer/SimpleFingerGestures_Android_Library
5. [NFC](https://stackoverflow.com/questions/23721036/how-to-combine-getintent-getaction-and-enableforegrounddispatch-on-same-class)
[https://stackoverflow.com/questions/23721036/how-to-combine-getintent-getaction-a](https://stackoverflow.com/questions/23721036/how-to-combine-getintent-getaction-and-enableforegrounddispatch-on-same-class)
[nd-enableforegrounddispatch-on-same-class](https://developer.android.com/guide/topics/connectivity/nfc/)
<https://developer.android.com/guide/topics/connectivity/nfc/>
6. [Acelerometro](https://code.tutsplus.com/es/tutorials/using-the-accelerometer-on-android--mobile-22125)
[https://code.tutsplus.com/es/tutorials/using-the-accelerometer-on-android--mobile-22](https://code.tutsplus.com/es/tutorials/using-the-accelerometer-on-android--mobile-22125)
[125](https://code.tutsplus.com/es/tutorials/using-the-accelerometer-on-android--mobile-22125)
7. [Dialog Flow](https://github.com/zoraidacallejas/Chatbot) <https://github.com/zoraidacallejas/Chatbot>