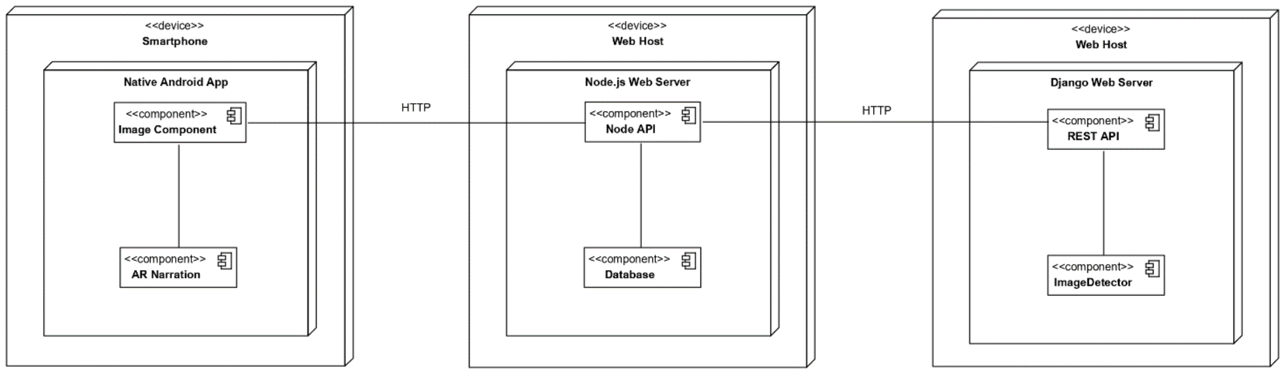
1. ARCHITECTURE OVERWIEV

At the hardware level, a distinction between three devices is made: the user's smartphones acts as side of the application: the Image component encapsulates the camera control and the network functionalities, while the AR Narration component is where the actual narration of the painting takes place.

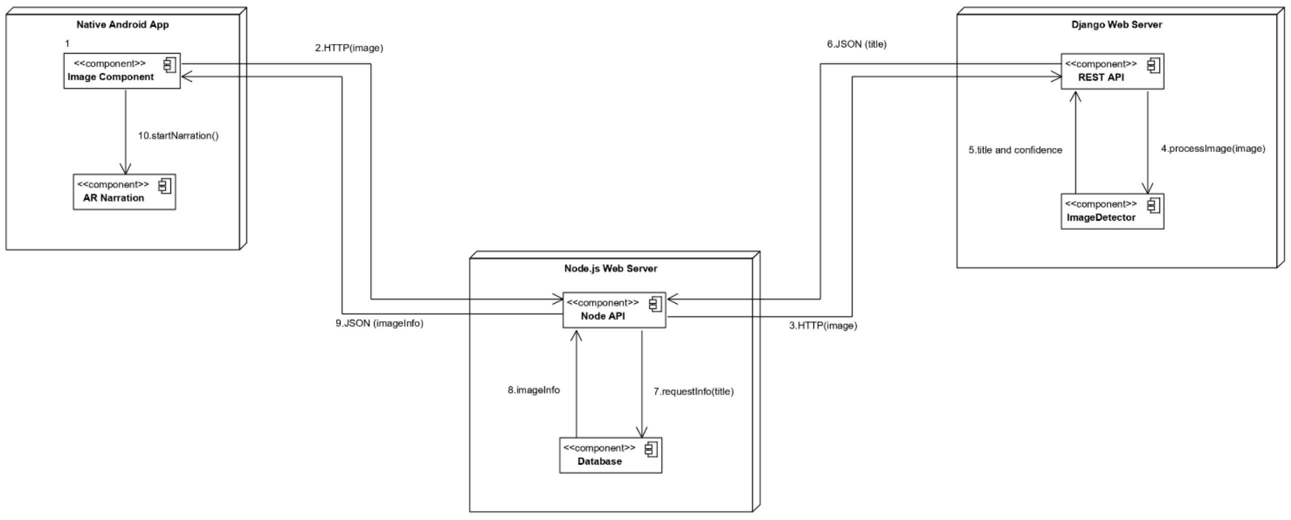
The Node.js server acts as a middle-ground between the client and the image recognition component. The Node API component manages the network requests, whil the Database component handles the persistend information associated to each painting.

A REST API, realized via the djangoRestFramework module, is used to interface the Django AI server with the Node server. Here the neural network is encapsulated in the ImageDetector component, in its own class which provides the methods necessary to run a pre-trained inference graph on the received image and output a title and a confidence score. If this confidence score, expressed as the probability of a specific painting being present in the scene, is greater than a pre-set threshold, a positive feedback is sent back to the Node server.

These two web servers could potentially be deployed onto the same machine; the reason they have been implemented as two distinct entities is just to keep a logic separation between the components.



1.1 IMAGE PROCESSING



Here a list of steps illustrating the processing of the image is presented:

1. User starts up the application and captures and image with the smartphone's integrated camera.
2. The captured image is sent to the Node.js server, using an HTTP multipart form-data request.
3. Once received, the image is forwarded from the Node API to Django server, via the exposed REST API.
4. The Django server receives the image and processes it via the underlying neural network, in the ImageDetector component.
5. A title and a confidence score are output for the received image by the neural network. It then passes it to the REST API component.
6. Response is then sent back to the Node.js server as a JSON object containing the two fields.
7. The Node.js server retrieves the information associated to the identified painting from a MongoDB relational database.
8. Any retrieved information is sent to the Node API component.
9. The information is sent back to the Android client.
10. The control is finally sent to the AR Component, which proceeds with the narration of the painting.

2. *ANDROID CLIENT*

The AR functionalities are provided by the ARCore SDK, Google's AR solution; when it comes to the handling of the 3D environment, the choice was between a fully-fledged game engine, like Unity or Unreal Engine, and an external Android library, such as Google's Sceneform. We decided for the latter, since this allowed us to keep a higher control over the whole application development; furthermore, this would make future maintenance or additions easier.

In order to handle the Text-To-Speech narration, the integrated android.speech.tts module has been used; since each TTS utterance is executed asynchronously, the narration works by setting up a custom UtteranceProgressListener for the TTS object and using two handlers to receive messages from the its thread during the utterance life cycle. The reason why the messaging is necessary in the first place, is because all operations influencing the augmented elements of the scene must be performed on the main thread on which the application is running.
The first handler, onTtsStartHandler, receives a message as soon as the speak() method on the TextToSpeech object has been called: it proceeds to generate the image corresponding to the segment which is being narrated and animates it forward in the environment. The second handler, onTtsDoneHandler, receives a message when an utterance has been successfully completed, without any errors, and it destroys the previously generated image node in the scene.

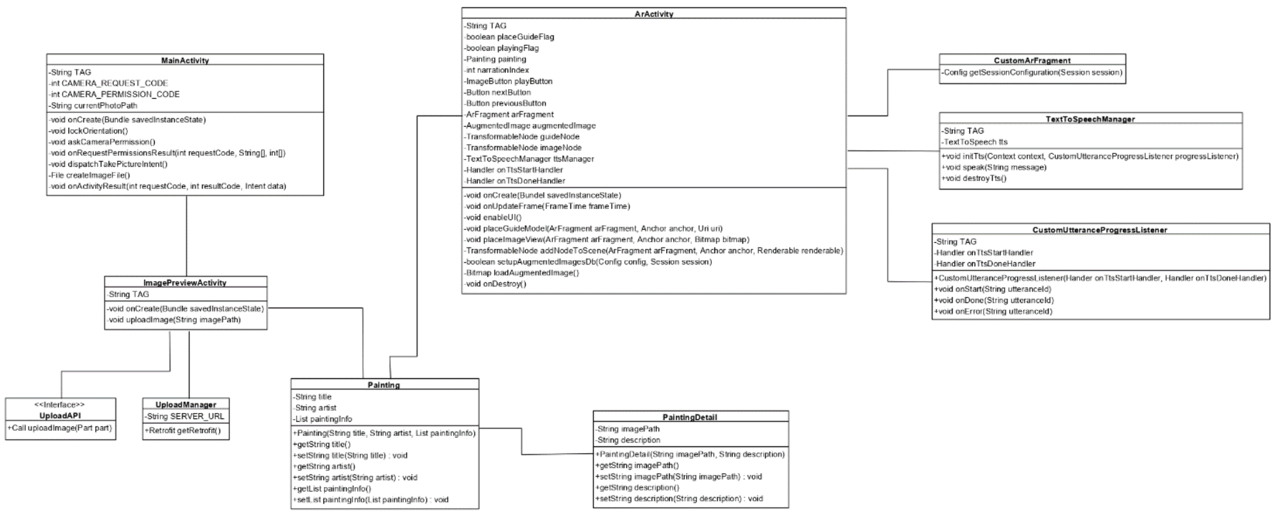
In order to provide a better user experience, we decided to implement user interface functionalities to navigate through the narration; these include a pause/resume button and two buttons to move to the previous and next narration segments. Since the native TTS module doesn't provide any pause functionalities we found ourselves with two main alternatives:
1. Make the TTS object record the narration to an audio file, using the native synthesizeToFile() method, and then use a MediaPlayer object to control the narration;
2. On resume, make the narration start from the beginning of the current segment.

By using the first approach any future implementation of some kind of interaction mechanisms between the user and the virtual guide would require a reengineering of the speech context. We decided to proceed with the latter, by carefully dividing the segments into smaller, independent parts.

Package structure:

```
App
│
├───────activities
│               ├───────MainActivity
│               ├───────ImagePreviewActivity
│               └───────ArActivity
│
├───────painting
│              ├───────Painting
│              └───────PaintingDetail
│
├───────texttospeech
│                ├───────TextToSpeechManager
│                └───────CustomUtteranceProgressListener
│
├───────network
│               ├───────UploadAPI
│               └───────UploadHandler
│
└───────CustomArFragment
```

Class diagram:


MainActivity
-String TAG
-int CAMERA_REQUEST_CODE
-int CAMERA_PERMISSION_CODE
-String currentPhotoPath
-void onCreate(Bundle savedInstanceState)
-void lockOrientation()
-void askCameraPermission()
-void onRequestPermissionsResult(int requestCode, String[], int[])
-void dispatchTakePictureIntent()
-File createImageFile()
-void onActivityResult(int requestCode, int resultCode, Intent data)
ImagePreviewActivity
-String TAG
-void onCreate(Bundle savedInstanceState)
-void uploadImage(String imagePath)
<<Interface>>
UploadAPI
+Call uploadImage(Part part)
UploadManager
-String SERVER_URL
+Retrofit getRetrofit()
ArActivity
-String TAG
-boolean placeGuideFlag
-boolean playingFlag
-Painting painting
-int narrationIndex
-ImageButton playButton
-Button nextButton
-Button previousButton
-ArFragment arFragment
-AugmentedImage augmentedImage
-TransformableNode guideNode
-TransformableNode imageNode
-TextToSpeechManager ttsManager
-Handler onTtsStartHandler
-Handler onTtsDoneHandler
-void onCreate(Bundel savedInstanceState)
-void onUpdateFrame(FrameTime frameTime)
-void enableUI()
-void placeGuideModel(ArFragment arFragment, Anchor anchor, Uri uri)
-void placeImageView(ArFragment arFragment, Anchor anchor, Bitmap bitmap)
-TransformableNode addNodeToScene(ArFragment arFragment, Anchor anchor, Renderable renderable)
-boolean setupAugmentedImagesDb(Config config, Session session)
-Bitmap loadAugmentedImage()
-void onDestroy()
CustomArFragment
-Config getSessionConfiguration(Session session)
TextToSpeechManager
-String TAG
-TextToSpeech tts
+void initTts(Context context, CustomUtteranceProgressListener progressListener)
+void speak(String message)
+void destroyTts()
CustomUtteranceProgressListener
-String TAG
-Handler onTtsStartHandler
-Handler onTtsDoneHandler
+CustomUtteranceProgressListener(Hander onTtsStartHandler, Handler onTtsDoneHandler)
+void onStart(String utteranceId)
+void onDone(String utteranceId)
+void onError(String utteranceId)
Painting
-String title
-String artist
-List paintingInfo
+Painting(String title, String artist, List paintingInfo)
+getString title()
+setString title(String title) : void
+getString artist()
+setString artist(String artist) : void
+getList paintingInfo()
+setList paintingInfo(List paintingInfo) : void
PaintingDetail
-String imagePath
-String description
+PaintingDetail(String imagePath, String description)
+getString imagePath()
+setString imagePath(String imagePath) : void
+getString description()
+setString description(String description) : void