Project EC327

Software Engineering

Prof. Densmore

Joel Akerman, Manuel Segimon, Axel Sariel, Jakub Zolkos, Taha Ababou

# 

Table of Contents

[Introduction: 2](#_Toc89879312)

[Server: 3](#_Toc89879313)

[*Figure 1* 4](file:////Users/joelakerman/Desktop/Project%20EC327%20Documentation.docx#_Toc89879314)

[*Figure 2* 4](file:////Users/joelakerman/Desktop/Project%20EC327%20Documentation.docx#_Toc89879315)

[*Figure 3* 5](file:////Users/joelakerman/Desktop/Project%20EC327%20Documentation.docx#_Toc89879316)

[*Figure 5* 6](file:////Users/joelakerman/Desktop/Project%20EC327%20Documentation.docx#_Toc89879317)

[*Figure 4* 6](file:////Users/joelakerman/Desktop/Project%20EC327%20Documentation.docx#_Toc89879318)

[Back-End: 6](#_Toc89879319)

[Front-End: 8](#_Toc89879320)

[UI Visual 9](#_Toc89879321)

[References: 9](#_Toc89879322)

# **Introduction:**

Our project consists in making an android application that would automatically capture pictures, name the object and also give their location. This app is intended to help visually impaired people so that they can know what is in front of them by just opening the app. Our UI is made in a way that the user doesn’t need to take the pictures. The app itself is going to take pictures and analyze them every couple of seconds by default.

# **Server:**

First, we had the dilemma on which API to use. Our options were Amazon Rekognition, IBM Watson Visual Recognition and Google Cloud Vision (which was the one we used). Why did we choose to use Google Cloud Vision over the others? First we ruled out IBM’s API because it costs money, then we had to make the decision between Google and Amazon. We choose Google because it was the easiest to implement because it allowed us to send the image directly from the server to the API while Amazon needed us to pass it from the server to one of their buckets before passing it to the API and then the response goes to the bucket again.

Then, we started to make the code for the image processing. The code starts by enhancing the picture so that Google’s API can have a better accuracy when detecting objects as shown in Figures 1 and 2. The way we enhanced the picture is by making it more dark that way the API can distinguish more between objects. To enhance the pictures we manipulated with their contrast, color, sharpness and brightness. We changed their values by a factor of 3, 0.75, 1.5,

0.75 respectively. We used these values after many tries of testing on over 20 images. After enhancing the image it saves the enhanced picture onto the server.



*Figure 1 – Before Enhancing*

*Figure 2 – Enhanced Picture with Google’s API*

We then need an object called “Object” which stores everything that we need about objects. We used our own object because Google’s one was really complicated and it also has extra data that we weren’t going to use. In our object (shown in figure 4) we have 3 functions: GetRelativePosition, HasGoodConfidence and GetName. GetRelativePosition is the function that is going to find where in the image the object is. This is done by setting the origin to the bottom left of the picture and then defining what should be left, right, center, etc. For our program we are going to be using left, slight left, center, slight right and right as our directions. We defined each location by an interval of 20% of the image from left to right. So from 0% – 20% will be left, 20% - 40% will be slight left, 40% - 60% will be center, 60% - 80% will be slight right and 80% - 100% is right (Figure 3). The program will identify the horizontal center of the object and the location of that center will be assigned as the location of the object. The next function inside “Object” is HasGoodConfidence. One of our main goals for our application is to have a really accurate app. That is exactly what this function will be doing. The function uses Google’s API to determine how accurate the prediction is. If the accuracy is lower than 50% then that object will not be counted as recognized. Because of our enhanced picture function we haven’t encountered a unknown object yet. The last one is GetName which as the name says it just uses Google’s API to get the name of the object recognized. A function that is outside the class but is still one the most important functions found in the same Objects file is GetObjects. GetObjects is the function that finds every single object that is recognized by Google’s API which is called with the FindFunctions function.

*Figure 3 -* What the program sees

For this project we are running a flask server. At first we all considered using an apache2 server where we would receive the image and call the python script through PHP, however we choose to use a flask server instead since it was the easiest way to implement our knowledge of programming and it would also be faster because it wouldn’t require to jump to PHP. The server first receives the image as a Base64 string and then it decodes it so that it could get a picture.





****

*Figure 5* – Main file

*Figure 4 – Object “Object”*

# **Back-End:**

****We developed the back-end of the Android app from its raw state and implementing various app functionalities. One of them was high frequency camera frame capture that was used to provide a stream of images sent as requests to the REST API and then Google Cloud Vision for analysis. Additionally, we implemented a function found in a third-party GitHub repository that converted the bitmaps created by the camera function to Base64 string format, which yielded a better response performance when fed to the server. Sending bitmaps directly would result in asynchronous response as it would require us to open many HTTP connections to send one image, making the image analysis longer and more tedious. Because of time constraints, we decided to stick with the real-time connection system between the app and the server that would send the processed frames capture by the camera to the server periodically with a fixed time interval of 5 seconds. This enabled us to omit the need for a button which somewhat reduces the complexity of app usage and could be favored more by a visually impaired user. I extrapolated the previously developed looping code to apply to the function sending the requests to the server. Because of Kotlin's syntax, it required a chain of nested function calls. While the camera function captures every frame and calls an image processing function for each of them, a function ConnectToServer() used to send requests to the server is invoked with one of those frames only once every few seconds. I also supplemented the same function with the text-to-speech engine that speaks out the string returned to the app by the server.

****We developed an Android Studio project that will be acting as a playground for the final app and also interfacing it with REST API. The main function made in the back-end was the scheduling one. The function executes the code every couple of seconds continuously. This makes the code refresh to seek for new objects that need to be recognized by the app showed in the camera. Another main functions was making the actual HTTP request to the REST API. We used the okhttp library in Kotlin to build this function. Most of the debugging was getting the request made correctly. The server had to be changed from HTTP to HTTPS a couple of times to get the security right, since we were having issues with the SSL certificates and trust settings in the app. In the end, we decided to stick to HTTP considering the time frame and requirements for the app. Also, getting the actual image sent to the server in an efficient way was somewhat challenging, since we had to juggle around different options to send it. We initially tried using the HTTP headers; that worked initially with a test string but then when the actual image was being sent, it was too big for a header. We moved to using the files section; this was working with our testing python scripts but we couldn’t get Kotlin to read the file and send it as part of the request, Finally we used the multipart form section in the request body to send the image, encoded in base64 format. In terms of the type of requests, we ended up using POST request to the server in order to send this data.

# **Front-End:**

The UI of this project was different from any work done by any of us in the past. Visually impaired people use their smartphones by having the phone voicing over on what is on the screen. That being said the UI must to be simple and crisp to use since our app is intended to be used by visually impaired people. Originally we wanted to make the app with extra features like bar-code and QR code scanning, between others. However, we realized that because of our intended audience, visually impaired people, we had to make the least amount of buttons possible. That is why there is no need to use any command just scan any environment with the phone.

To launch the splash screen we used Java and Kotlin to. The splash screen includes the logo and app name. The splash screen helps the app load properly in the background. The splash screen takes 2 seconds long which is the optimal amount of time to make the app load properly and also making it not look as the app froze. After the splash screen there is the frame layout that includes the camera created in the Back-End of the app.

****

## **UI Visual**

# **References:**

Third Party Camera Code:

<https://github.com/hamza372/FramesOfLiveFeedKotlin>

Google’s API:

<https://cloud.google.com/vision>

****