DIY Mirror

Xiaozhou Liang, Hongwei Wu
Department of Computer Science
SUNY Binghamton
Binghamton, NY, USA
{xliang24,hwu71}@binghamton.edu

ABSTRACT

MagicMirror^2 is a smart mirror that supports abundant third-party modules and provides users with convenient services. However, it has not supported the use scenario of multiple users with different service demands. To amend this deficiency, we introduce face recognition and user interface customization to enable our users to customize their mirrors. We build this *DIY Mirror* project, design a system composed of three main components, and implement the functionality. To evaluate our *DIY Mirror*, we carry out a demonstration and prove the successful operations of our system.

KEYWORDS

MagicMirror^2, Face Recognition, Raspberry Pi, Android Application, Google Cloud Platform, Google Assistant

1 INTRODUCTION

In this hi-tech era, time is money and people look for an efficient way to save their time. We have always wanted to try to condense people's daily routines into concurrent operations to help people save more time, until we found that a GitHub open source project *MagicMirror*^{2}[1] can help achieve this purpose, by displaying some helpful information on the screen behind the mirror. However, the existing Magic Mirror does not support multi-user customized display. Therefore, we would like to focus on making further improvements on the existing *MagicMirror*^{2} project, by introducing face recognition, voice control and an Android application, to meet the needs of user customization. Therefore, we call our project *DIY Mirror*.

2 BACKGROUND

In this section, we first discuss the software for this project, followed by a brief introduction of the hardware components.

2.1 Software

2.1.1 MagicMirror^2. *MagicMirror*^{2}[7] is an open-source modular smart mirror platform on GitHub. Built by the same author, Micheal Teeuw, *MagicMirror*^{2} is a upgraded version of the original *MagicMirror* that support modularization. Developed by the *MagicMirror*^{2} community, modules are third-party services that can be added to *MagicMirror*^{2}, with examples including weather forecasting and news broadcasting. With the help of *MagicMirror*^{2} and abundant installable modules, users are able to combine a mirror with a screen monitor, and turn them into a helpful personal assistant. By default, there is only one config.js file in the project, and it does not support multi-user customized module configuration. In our project, *MagicMirror*^{2} serves as a foundation project. And our goal is to enable multiple users to customize their module configuration, and use voice command as well as face recognition to adjust the display content corresponding to the users. We forked[2] the original *MagicMirror*^{2} project and added our extended implementation including face recognition, voice control, configuration generation, etc.

2.1.2 OpenCV. OpenCV[1] (Open Source Computer Vision Library) is an open-source BSD-licensed library that includes several hundreds of computer vision algorithms. In our project, we use OpenCV to facilitate face recognition process, which is executed on the Raspberry Pi in Python.

2.1.3 Google Assistant. Google Assistant is a virtual assistant that supports two-way conversions developed by Google. Compared to its competitor Amazon Alexa, Google Assistant can handle free-formed, web-based queries better[6]. Supporting various operating systems including Raspberry Pi and Android, it provides developers with a software development kit (SDK) that supports features such as setting customized actions. In our project, we register the Raspberry Pi as a hardware device of a Google Assistant project and utilize its voice control to trigger the Pi Camera.

2.1.4 Google Compute Engine. Google Compute Engine is an Infrastructure as a Service (IaaS) component of Google Cloud Platform. Preferable for a lower price, Google Compute Engine has attracted companies including Spotify, Snapchat, etc[4]. Users can launch general-purpose virtual machines on it, and by setting a static IP address, it becomes accessible from the Internet. To serve our *DIY Mirror* project, We deploy a server on Google Compute Engine, which is referred to as Cloud Server in this paper.

2.1.5 Google Cloud SQL Database. As another service provided by Google Cloud Platform, Google Cloud SQL supports relational database creation and management. Similar to Google Compute Engine, by setting up a publicly accessible IP address, it can be constructed as a database service supporting transactions via this address. To store structured data involved in our *DIY Mirror* project, we set up a MySQL database on Google Cloud SQL, which is referenced as Cloud Database below.

2.1.6 Android Studio. As the official integrated development environment (IDE) for Android operating system, Android Studio is a popular tool for Android Application development. It provides user-friendly features including real-time preview and emulators, which boost developers' productivity and relieve the effort of development[3]. Following the principle of separation of concerns, the Android Application development is split into the front-end

[1]https://github.com/MichMich/MagicMirror

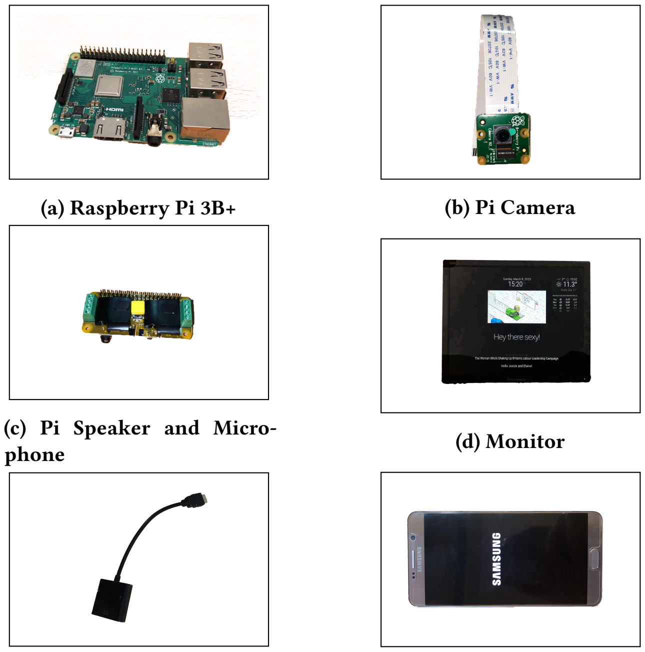
[2]https://github.com/hwu71/MagicMirror

XML-based layout implementation and the back-end implementation in Java or Kotlin. In this project, we use Android Studio to develop an Android Application with Java back-end implementation that supports UI customization for our *DIY Mirror*.

2.2 Hardware

We utilize and install the following hardware components in this project (as shown in Figure 1):

- Raspberry Pi 3B+: Raspbian Buster
- Pi Camera: Raspberry Pi Camera Module V2-8 Megapixel, 1080p
- Pi Speaker and Microphone: Raspiaudio MIC+
- Monitor: Lenovo LT1913pA 19” 1280x1024 Flat Panel LCD Monitor Grade B
- HDMI to VGA Adapter
- Android Phone: Samsung Galaxy Note 5



(a) Raspberry Pi 3B+ **(b) Pi Camera** **(c) Pi Speaker and Microphone** **(d) Monitor** **(e) HDMI to VGA Adapter** **(f) Android Phone**

Figure 1: Hardware Components

3 DESIGN

As shown on figure 2, there are 3 main components in our architecture design: Google Cloud Platform, Raspberry Pi and affiliated hardware components and the Android Application. We will discuss each of them separately in this section.

3.1 Google Cloud Platform

There’re several services deployed on the Google Cloud Platform. They are the Cloud Server, the Cloud Database and the Google Assistant project.

1. Cloud Server. On the Cloud Server, we run the server-side code of *MagicMirror*^{2}, as well as a TCP server to handle requests of reloading the configuration file for a user so that the *DIY Mirror* could display customized UI.

2. Cloud Database. We also host a MySQL database on the Google Cloud SQL. The Android Application establish a connection to the Cloud Database to read and write user account information, as well as the module configuration information. Besides, the database is read to generate the user customized configuration file of the *DIY Mirror* on the Cloud Server.

3. Google Assistant Project. We build a Google Assistant project managed by the Actions Console, which utilizes the Google Assistant API to fulfill the voice control functionality. We register the Raspberry Pi as a hardware device of the Google Assistant project, so that Pi is able to interact with this Google Assistant project. Thus, when user says a "Face recognition" command to the Raspberry Pi, our Google Assistant project is able to recognize the voice command and execute the corresponding shell script to trigger the face recognition program.

3.2 Raspberry Pi and Affiliated Hardware Components

This part consists of the Raspberry Pi, the Pi Microphone and Speaker, the Pi Camera, and the Monitor. It carries the following functionalities:

1. Face Recognition. We run the OpenCV-based Face Recognition program on Raspberry Pi with Pi Camera. The Face Register program will be triggered by the Android Application at the time when user registration. At the time of common using, the Face Recognition Program will be triggered by voice command through the Google Assistant, and send the user identity (in this case, the registered username) to the Cloud Server after recognizing the user.

2. Voice Control. We added the Voice Control function using the Pi Microphone and Speaker, with the help of the Google Assistant project. Users are able to use the on-board Pi Microphone and Speaker to interact with Google Assistant, to help trigger the Face Recognition program. This function can be further extended to other needs such as playing some music, etc.

3. TCP Server. We run a TCP server on Raspberry Pi to handle incoming requests. It classifies the type of message according to its header. There are two types of headers: `PHOTO_COLLECTION` and `REFRESH`. Messages with the former header would trigger the Python script for photo collection and consequent dataset training. Messages with the latter would trigger the Python script for refreshing the *DIY Mirror* so as to update the UI.

4. Monitor display. Our Raspberry Pi is connected to the Screen Monitor via an HDMI cable. The Monitor will display the web page rendered by the server-side code of the *MagicMirror*^{2} running on the Cloud Server.

3.3 Android Application

The Android Application will interact with the other two parts as well. When user login the Application for the first time, he or she will need to create a new account and press the camera button on the phone screen to trigger the Face Registration process on the Raspberry Pi. At the time of common using, users can customize

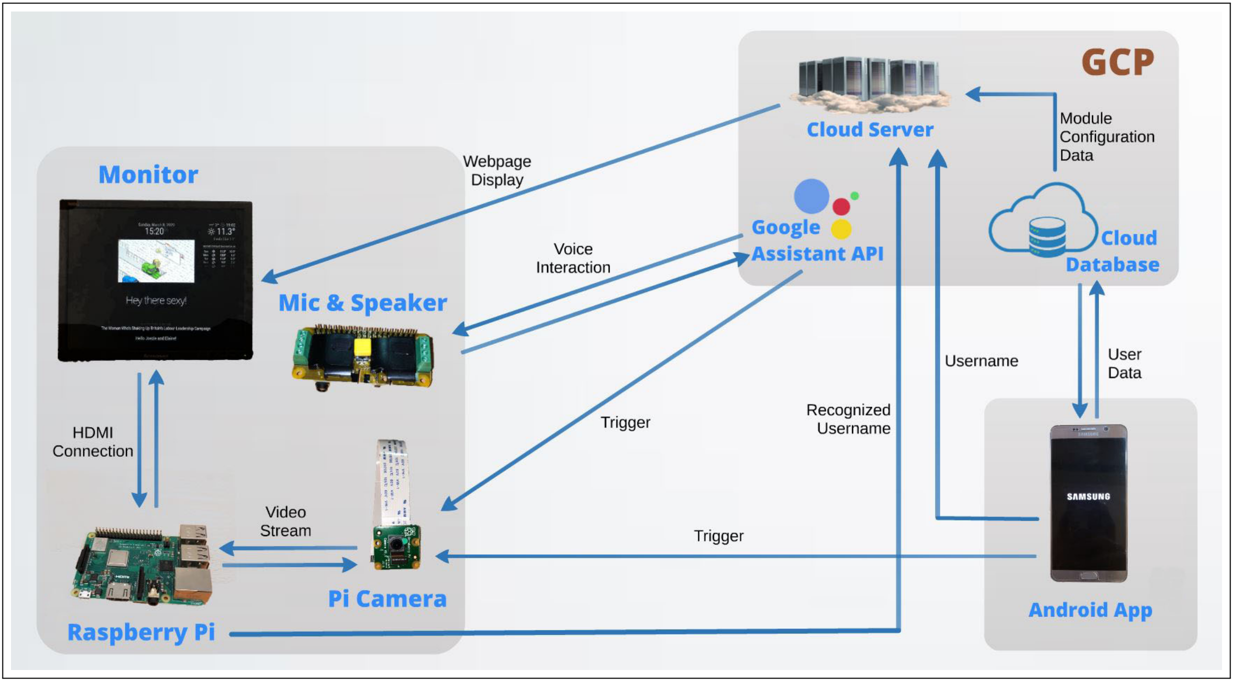


Figure 2: Architecture Design

the User Interface of the *DIY mirror*, and to set the positions and properties of the modules that they want.

4 IMPLEMENTATION

In this section, we will describe the implementation in details from four perspectives. They are Face Recognition, Voice Control, Cloud Database and Android Application.

4.1 Face Recognition

In order to provide the user customization functionality in our *DIY Mirror* project, we introduced the Face Recognition based on OpenCV[5]. The Face Recognition operation is consist of three components.

Part 1: Collect the face dataset. When a user registers him/herself on the Android Application and presses the camera button, a message containing the username will be sent from the Android App to the Raspberry Pi . After the Raspberry Pi receives this message, it will start a Python script to start the Pi Camera and take 30 pictures of the user, and store them under the dataset folder with the username as the name of the sub-folder.

Part 2: Train the dataset. Each time after a new user's face photos are added to the dataset, we will need to re-compute the face embedding, which will be used as a known-database in later face recognition in Part 3.

Part 3: Recognize faces in video streams. When the user says the "face recognition" voice command to the Pi Microphone, a Python script that executes the actual face recognition program will be triggered. After activating the Pi Camera, this program gets the video stream from it, captures images of the user's face, and matches it with the known-database(face embedding) generated in Part 2. Each comparison returns the most similar user's name. Since the accuracy of the face recognition based on OpenCV is not yet 100%, we take the strategy to perform multiple (10 times) comparisons and return the results with the highest frequency to improve accuracy. After recognizing the user's identity, a trigger message containing the recognized username will be sent from the Raspberry Pi to the TCP server running on the Cloud Server, triggering generation of a new config.js file according to the user's module configuration information stored on the Cloud Database. On success, a refresh message will be sent from the Cloud Server to the TCP server running on the Raspberry Pi and consequently refresh the *DIY Mirror*. Finally, the *DIY Mirror* would get updated and reflect the identity of the user.

4.2 Voice Control

To trigger the Face Recognition process, we utilized the on-board Pi Speaker and Microphone, with the help of the Google Assistant Service[2] from Google Cloud Platform. Following shows 4 steps of our implementation.

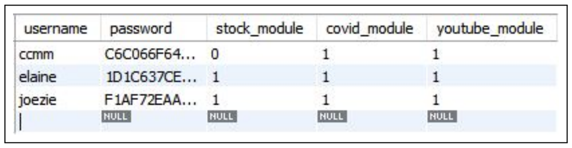
Step 1: Configure a developer project and account settings. As the first step, we created a Google Cloud Platform project managed by the Actions Console, which will have access to the Google Assistant API. In the Actions Console, we can fill in the register information about the hardware devices, edit the invocation key word, build the actions of this project, and test the built actions, etc.

Step 2: Register device. First, we filled in the device information in the actions console using the registration UI. Then we downloaded the OAuth credentials (a JSON file), and sent it to the hardware device, the Raspberry Pi in our case, for later registration. Then on the Raspberry Pi, after downloading the prerequisite packages and setting the virtual environment, we used the authorization tool `google-oauthlib-tool` with the credential files to complete the registration process of our Raspberry Pi. This hardware registration process will require visiting a website and returning a given code to the authorization tool.

Step 3: Design the voice actions. In the Actions Console, we added a new action named Face Recognition, which will recognize the voice command like "Do face recognition, please" or just "Face recognition", and tell the user that it is performing the face recognition operation. Besides, we also modified the original welcome action to explicitly tell the user that it is a greeting from the *DIY Mirror* project.

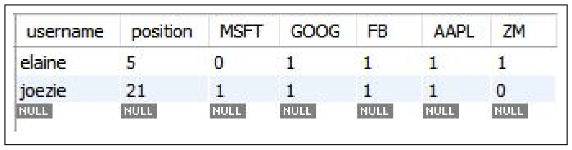
Step 4: Implement functionality. After cloning the sample GitHub project for Google Assistant API[3], we modified the pushtotalk.py file by adding a string matching with the returned text output from the Face Recognition action. If it contains the phrase “face recognition”, we will trigger a Python script to execute the face recognition process in the Raspberry Pi.

4.3 Cloud Database



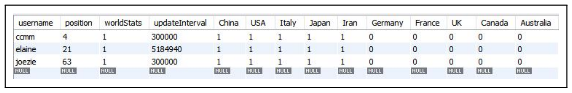
username	password	stock_module	covid_module	youtube_module
ccmm	C6C066F64...	0	1	1
elaine	1D1C637CE...	1	1	1
joezie	F1AF72EAA...	1	1	1
|	NULL	NULL	NULL	NULL

(a) User-Module Table



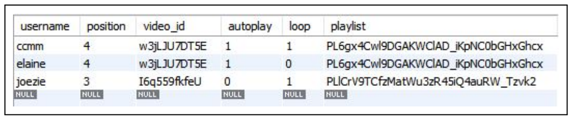
username	position	MSFT	GOOG	FB	AAPL	ZM
elaine	5	0	1	1	1	1
joezie	21	1	1	1	1	0
NULL	NULL	NULL	NULL	NULL	NULL	NULL

(b) Stock Module Configuration Table



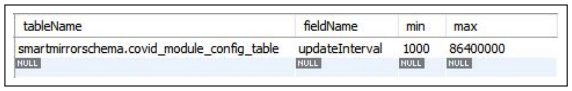
username	position	worldStats	updateInterval	China	USA	Italy	Japan	Iran	Germany	France	UK	Canada	Australia
ccmm	4	1	300000	1	1	1	1	1	0	0	0	0	0
elaine	21	1	5184940	1	1	1	1	1	0	0	0	0	0
joezie	63	1	300000	1	1	1	1	1	0	0	0	0	0
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

(c) Covid Module Configuration Table



username	position	video_id	autoplay	loop	playlist
ccmm	4	w3jLJU7DT5E	1	1	PL6gx4Cwl9DGAKWClAD_iKpNC0bGHxGhcx
elaine	4	w3jLJU7DT5E	1	0	PL6gx4Cwl9DGAKWClAD_iKpNC0bGHxGhcx
joezie	3	I6q559fkfeU	0	1	PLlCrV9TCfzMatWu3zR45iQ4auRW_Tzvk2
NULL	NULL	NULL	NULL	NULL	NULL

(d) YouTube Module Configuration Table



tableName	fieldName	min	max
smartmirrorschema.covid_module_config_table	updateInterval	1000	86400000
NULL	NULL	NULL	NULL

(e) Range Table

Figure 3: Cloud Database Tables

The Cloud Database is a “bridge” between the Cloud Server and the Android Application, considering that it stores the user information and module configuration data provided by the Android Application and consumed by the Cloud Database. Figure 3 shows multiple tables created in the database for various purposes. The user-module table in figure 3a stores usernames, encrypted passwords, and enabling statuses of each module, where 0 represents status “disabled”, and 1 represents status “enabled”. It is noteworthy that by encrypting the passwords when storing and decrypting when matching, we are able to prevent privacy theft even in the extreme case that the database information is exposed. Figure 3b, 3c, and 3d are examples of module configuration tables, where configuration properties and the positions on the *DIY Mirror* are stored as separate fields. By separating module information into different tables, we make it extensible to support new modules by merely creating new module tables for them. The range table in Figure 3e stores the ranges of integer-type field from other tables. On our Android Application, we provide seek bars where users should choose integer values within the given range read from this table.

4.4 Android Application

To support users’ customization of the user interface (UI) of the *DIY Mirror*, we created a GitHub project[4] and developed an Android Application targeted for Android 7.0 (Nougat) using Android Studio version 3.6. As shown in Figure 4, the main functions of this application include user registration, user login, UI overview, and module configuration update. More specifically, at the time of registration, users create a new account and take photos of their faces. At the time of common using, users can customize UI of the *DIY Mirror* including which modules to display, as well as the positions and properties of modules.

As a component in our system, the Android Application communicates with other components to provide all-around services to our users. The Cloud Database is the component that shares the most frequent communication with the Android Application. At the time of registration and user login, users’ username and password information are stored into and read from the User-Module table. Besides, to display the UI overview of all positions on the *DIY Mirror*, we also need information from various tables of module configuration tables. Last but not least, as the key services, the adding and removing operations on modules, as well as updating operations on module configurations, require reading and updating multiple tables in the database.

The Android Application also communicates with the TCP servers running on the Raspberry Pi and on the Cloud Server. Triggered by pressing the camera button during registration, a message is sent to the Raspberry Pi which then activates the Pi Camera to take photos of the user’s face. Besides, when adding, removing, or updating modules, it also send a message to the Raspberry Pi and the Cloud Server respectively. If the user is using the *DIY Mirror*, then the messages would trigger the page refresh script on the Raspberry Pi and the configuration file generation script on the Cloud Server.

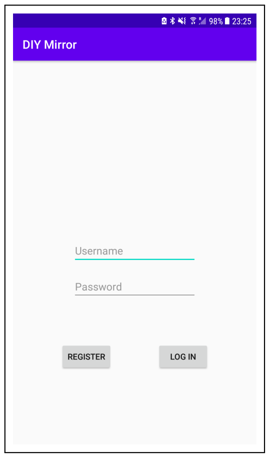
5 EVALUATION

A walk through of our *DIY Mirror* project can be seen in our demo video. We will describe the demo process step by step in this section.

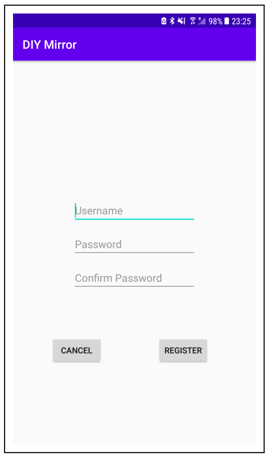
Step 1. At the beginning when no user is logged in, the *DIY Mirror* should show "Hello, stranger!".

[3]https://github.com/googlesamples/assistant-sdk-python

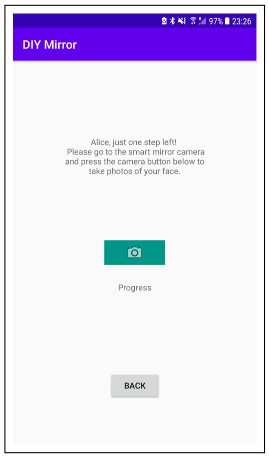
[4]https://github.com/joezie/SmartMirrorModuleCustomizationApp



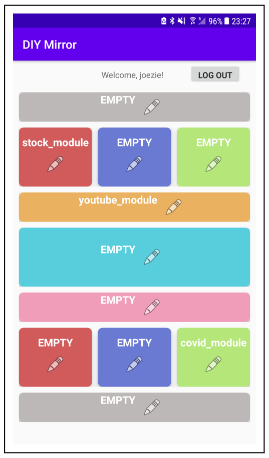
(a) Login Page



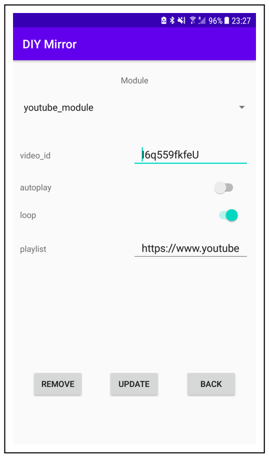
(b) Register Page


DIY Mirror
Alice, just one step left!
Please go to the smart mirror camera
and press the camera button below to
take photos of your face.
Progress
BACK

(c) Photo Collection Page



(d) UI Overview Page



(e) Module Update Page

Figure 4: Android Application Graphical User Interface Design

Step 2. When a new user registers himself at the Android Application, this operation should trigger the Pi Camera to take several pictures of the user, serving as the training set for consequent face recognition purposes. After the photo collection is completed and the photos data is successfully trained, the user should receive a notification on the Android Application.

Step 3. After the new user has registered on the Android Application, he/she should be able to use voice command to trigger Face Recognition to show a default *DIY Mirror* display with his/her username on it.

Step 4. The user is able to log in on the Android Application and update the module configuration. The *DIY Mirror* display should be automatically refreshed after the update operation, with the newly-updated modules' configuration.

Step 5. In a multi-user scenario, when an already-registered user uses the voice command to trigger the Face Recognition, the *DIY Mirror* should be able to display the UI corresponding to the user's customized module configuration information.

Through our demonstration, all five requirements are met, proving the correctness of the *DIY Mirror* project.

6 CONCLUSION

DIY Mirror solves the problem of *MagicMirror*^{2}, that is, only one module configuration file is supported, thus multiple users cannot customize their own module display. We solve the problem by implementing an Android Application for the user to set the personal module configuration, as well as by introducing Face Recognition and Voice control. In this case, multiple users only need to register their faces and fill in their personal module preferences in the Application, then by saying "Face Recognition" to the Mirror and wait for a while for the face recognition, each user can easily request the Mirror to display their customized Module content.

REFERENCES

[1] G. Bradski. 2000. The OpenCV Library. *Dr. Dobb's Journal of Software Tools* (2000).

[2] Google. 2020. *Introduction to the Google Assistant Service.* https://developers.google.com/assistant/sdk/guides/service/python [Online; accessed 3-May-2020].

[3] Zach Honig. 2013. *Google intros Android Studio, an IDE for building apps.* https://www.engadget.com/2013-05-15-google-android-studio.html [Online; accessed 2-May-2020].

[4] Brian Jackson. 2020. *Top 7 Advantages of Choosing Google Cloud Hosting.* https://kinsta.com/blog/google-cloud-hosting/ [Online; accessed 2-May-2020].

[5] Adrian Rosebrock. 2018. *Raspberry Pi Face Recognition.* https://www.pyimagesearch.com/2018/06/25/raspberry-pi-face-recognition/ [Online; accessed 3-May-2020].

[6] Sascha Segan. 2019. *Amazon Echo vs. Google Home: Which Smart Speaker Is Best?* https://www.pcmag.com/news/amazon-echo-vs-google-home-which-smart-speaker-is-best [Online; accessed 2-May-2020].

[7] Michael Teeuw. 2020. *MagicMirror*^{2} *Documentation.* https://docs.magicmirror.builders/ [Online; accessed 3-May-2020].