

Winter 2019

Smart Doorbell Camera

Project Book

**Performed by:**

Asaf Amnony, Guy Abramovich

**Instructor:**Oren Kalinsky

Table of Contents

[3 Abstract 4](#_Toc17215009)

[4 Project Specifications 5](#_Toc17215010)

[4.1 System Properties 5](#_Toc17215011)

[4.2 System Concept 5](#_Toc17215012)

[5 Hardware Components 6](#_Toc17215013)

[5.1 Raspberry Pi 6](#_Toc17215014)

[5.2 Raspberry Pi Camera 7](#_Toc17215015)

[5.3 Android-Based Smartphone 8](#_Toc17215016)

[6 Cloud Services 9](#_Toc17215017)

[6.1 Firebase Cloud Messaging 9](#_Toc17215018)

[6.2 Amazon Web Services 10](#_Toc17215019)

[6.2.1 Rekognition 10](#_Toc17215020)

[6.2.2 IOT 10](#_Toc17215021)

[6.2.3 EC2 11](#_Toc17215022)

[6.2.4 S3 11](#_Toc17215023)

[6.2.5 DynamoDB 12](#_Toc17215024)

[6.2.6 Cognito 12](#_Toc17215025)

[6.2.7 IAM 12](#_Toc17215026)

[6.2.8 ACM 13](#_Toc17215027)

[7 Additional Software Libraries 14](#_Toc17215028)

[7.1 OpenCV 14](#_Toc17215029)

[7.2 UV4L 14](#_Toc17215030)

[7.3 Jitsi 15](#_Toc17215031)

[8 System Architecture 16](#_Toc17215032)

[9 Raspberry Pi Detailed Design 18](#_Toc17215033)

[9.1 Required External Packages 19](#_Toc17215034)

[9.2 Electrical Circuit 19](#_Toc17215035)

[10 EC2 Detailed Design 20](#_Toc17215036)

[11 Android Detailed Design 21](#_Toc17215037)

[11.1 LoginActivity 22](#_Toc17215038)

[11.2 MenuActivity 22](#_Toc17215039)

[11.3 LogActivity 23](#_Toc17215040)

[11.3.1 MsgAdapter and TableRowModel 23](#_Toc17215041)

[11.4 MessagesActivity 24](#_Toc17215042)

[11.5 MQTT 24](#_Toc17215043)

[11.6 StreamActivity 25](#_Toc17215044)

[11.7 FcmNotification 26](#_Toc17215045)

[12 Issues and Resolutions 27](#_Toc17215046)

[13 Future Tasks 28](#_Toc17215047)

[14 Bibliography 29](#_Toc17215048)

Table of Figures

[Figure 1 - Smart Home 4](#_Toc17215049)

[Figure 2- System Concept 5](https://technionmail-my.sharepoint.com/personal/amnonya_campus_technion_ac_il/Documents/winter1819/044170_project_c/project_book.docx#_Toc17215050)

[Figure 3 - The System 5](#_Toc17215051)

[Figure 4 - Raspberry Pi 3 Model B 6](#_Toc17215052)

[Figure 5 - Raspberry Pi Logo 6](#_Toc17215053)

[Figure 6 - Raspberry Pi Camera Module v2.1 7](#_Toc17215054)

[Figure 7 - Android Logo 8](#_Toc17215055)

[Figure 8 - FCM Logo 9](#_Toc17215056)

[Figure 9 - AWS Rekognition Logo 10](#_Toc17215057)

[Figure 10 - AWS IOT Logo 10](#_Toc17215058)

[Figure 11 - AWS EC2 Logo 11](#_Toc17215059)

[Figure 12 - AWS S3 Logo 11](#_Toc17215060)

[Figure 13 - AWS DynamoDB Logo 12](#_Toc17215061)

[Figure 14 - AWS Cognito Logo 12](#_Toc17215062)

[Figure 15 - AWS IAM Logo 12](#_Toc17215063)

[Figure 16 - AWS ACM Logo 13](#_Toc17215064)

[Figure 17 - OpenCV Logo 14](#_Toc17215065)

[Figure 18 - Jitsi Logo 15](#_Toc17215066)

[Figure 19 - System Architecture 16](https://technionmail-my.sharepoint.com/personal/amnonya_campus_technion_ac_il/Documents/winter1819/044170_project_c/project_book.docx#_Toc17215067)

[Figure 20 - Raspberry Pi Finite State Machine 18](#_Toc17215068)

[Figure 21 - RP Electrical Circuit 19](#_Toc17215069)

[Figure 22 - App Overview 21](#_Toc17215070)

[Figure 23 – LoginActivity 22](#_Toc17215071)

[Figure 24 - MenuActivity 22](#_Toc17215072)

[Figure 25 - LogActivity 23](#_Toc17215073)

[Figure 26 - MessagesActivity 24](#_Toc17215074)

[Figure 27 - StreamActivity 25](#_Toc17215075)

[Figure 28 - FcmNotification 26](#_Toc17215076)

# Abstract

The concept of home automation has been around for around 100 years, from the early beginnings of electrical appliances such as basic washing machines, dishwashers, etc. to Artificial Intelligence and Internet of Things which let appliances communicate with themselves.

Homes are becoming smarter and smarter!

One aspect of smart homes is security, mainly door security – granting access only to authorized personal and continuously logging every action which homeowners can view at any time.

This project integrates software and hardware to develop a mobile application for Android devices, which provides users the ability to monitor and control a smart door which opens itself for recognized personal.



Figure 1 - Smart Home

# Project Specifications

## System Properties

* The system shall open the door when a recognized person arrives at the door
* The system shall notify the owner with an Android app notification and a snapshot when an unrecognized person rings the doorbell
* The owner shall be able to open the door
* The owner shall be able to view a livestream of the door
* The owner shall be able to view a log of all activities of the system

## System Concept

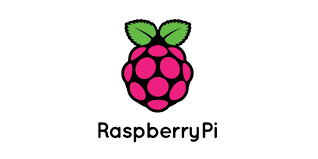


Figure 2- System Concept

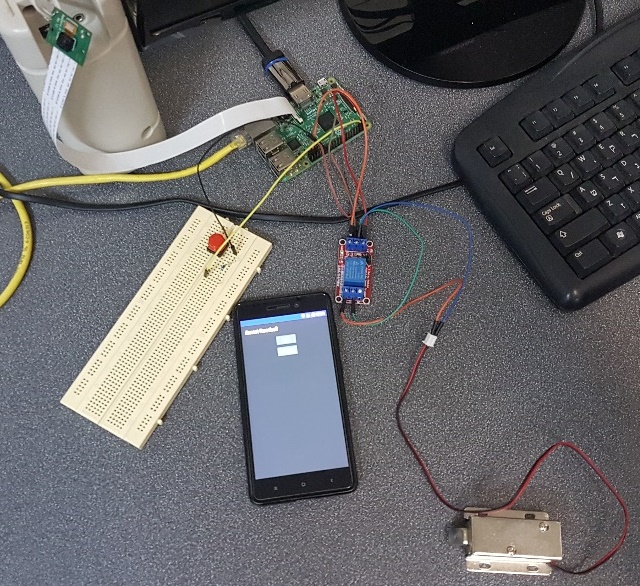


Figure 3 - The System

# Hardware Components

## Raspberry Pi

The Raspberry Pi (RP) is a series of small single-board computers.

In this project, we used Raspberry Pi 3 Model B [Size: 85.60 mm × 56.5 mm × 17 mm].



Figure 4 - Raspberry Pi 3 Model B

Specification:

* CPU - 1.2 GHz 64-bit quad-core ARM Cortex-A53
* Memory – 1GB
* Four USB ports
* 15-pin MIPI camera interface (CSI) connector (used with the Raspberry Pi camera)
* HDMI output
* 10/100 Mbit/s Ethernet (8P8C) USB adapter
* GPIO (General purpose input-output) 40 pins layout
* GPU - Broadcom VideoCore IV @ 250 MHz
* Operating systems - Raspbian, a Debian-based Linux operating system
* Cost – ~ 35$

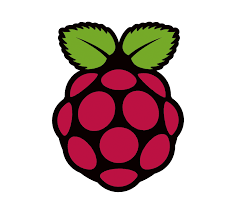


Figure 5 - Raspberry Pi Logo

## Raspberry Pi Camera

The Raspberry Pi Camera Module is an official product from the Raspberry Pi Foundation.

The original 5-megapixel model was released in 2013, and an 8-megapixel Camera Module v2 was released in 2016.

In this project, we used Camera Module v2.1.



Figure 6 - Raspberry Pi Camera Module v2.1

Specification:

* Size - 25 × 24 × 9 [mm]
* Weight - 3 [gram]
* Still resolution - 8 [Megapixels]
* Video modes - 1080p30, 720p60 and 640 × 480p60/90
* Sensor - Sony IMX219
* Video formats - raw h.264 (accelerated)
* Cost - ~ $25

## Android-Based Smartphone

The application was implemented on Android operating system.

Android is a mobile operating system developed by Google, based on a modified version of the Linux kernel and other open source software and designed primarily for touchscreen mobile devices such as smartphones and tablets.

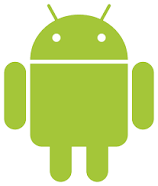


Figure 7 - Android Logo

# Cloud Services

## Firebase Cloud Messaging

Firebase Cloud Messaging (FCM), formerly known as Google Cloud Messaging (GCM), is a cross-platform cloud solution developed by Google, for messages and notifications for Android, iOS, and web applications.

In this project, the communication from the Raspberry Pi to the Android application uses FCM push notifications.



Figure 8 - FCM Logo

## Amazon Web Services

Amazon Web Services (AWS) are a major component in this project and contain vast amount of services that were used, which are described in the following sections.

### Rekognition

AWS Rekognition makes it easy to add image and video analysis to your applications. You just provide an image or video to the Rekognition API, and the service can identify the objects, people, text, scenes, and activities, as well as detect any inappropriate content. AWS Rekognition also provides highly accurate facial analysis and facial recognition on images and video that you provide. You can detect, analyze, and compare faces for a wide variety of user verification, people counting, and public safety use cases.

AWS Rekognition is based on the same proven, highly scalable, deep learning technology developed by Amazon’s computer vision scientists to analyze billions of images and videos daily and requires no machine learning expertise to use.

In this project, AWS Rekognition is used to analyze and detect authorized personnel faces.

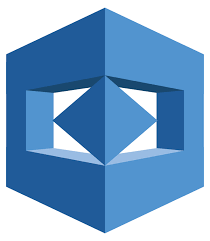


Figure 9 - AWS Rekognition Logo

### IOT

The Internet of things (IoT) is the network of physical devices, vehicles, home appliances and other items embedded with electronics, software, sensors, actuators, and network connectivity, which enables these objects to connect and exchange data.

Through AWS IOT, few components were registered:

* Raspberry Pi
* AWS EC2 instance
* Android application

These components can communicate with each other after registration.

In this project, the communication from the Android application to the Raspberry Pi is uses AWS IOT alongside the MQTT protocol:

MQTT (MQ Telemetry Transport or Message Queuing Telemetry Transport) is an ISO standard (ISO/IEC PRF 20922) publish-subscribe-based messaging protocol. It works on top of the TCP/IP protocol. It is designed for connections with remote locations where a "small code footprint" is required or the network bandwidth is limited. The publish-subscribe messaging pattern requires a message broker.



Figure 10 - AWS IOT Logo

### EC2

Amazon Elastic Compute Cloud (AWS EC2) provides scalable computing capacity in the Amazon Web Services (AWS) cloud. Using AWS EC2 eliminates your need to invest in hardware up front, so you can develop and deploy applications faster. You can use AWS EC2 to launch as many or as few virtual servers as you need, configure security and networking, and manage storage. AWS EC2 enables you to scale up or down to handle changes in requirements or spikes in popularity, reducing your need to forecast traffic.

In this project, an AWS EC2 instance is used to run the video stream server (Jitsi) with no hardware dependency.

In that way, the video stream can be reached from anywhere and can communicate with other relevant AWS services.



Figure 11 - AWS EC2 Logo

### S3

Amazon Simple Storage Service is storage for the Internet. It is designed to make web-scale computing easier for developers.

AWS S3 has a simple web services interface that you can use to store and retrieve any amount of data, at any time, from anywhere on the web. It gives any developer access to the same highly scalable, reliable, fast, inexpensive data storage infrastructure that Amazon uses to run its own global network of web sites. The service aims to maximize benefits of scale and to pass those benefits on to developers.

In this project, AWS S3 service is used to store images captured by the Raspberry Pi camera during major events that occur in the system, for log purposes, and images of recognized and authorized personnel.

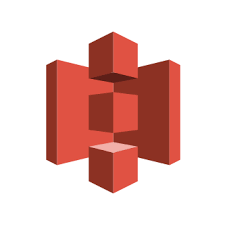


Figure 12 - AWS S3 Logo

### DynamoDB

AWS DynamoDB is a key-value and document database that delivers single-digit millisecond performance at any scale. It's a fully managed, multiregion, multimaster database with built-in security, backup and restore, and in-memory caching for internet-scale applications. DynamoDB can handle more than 10 trillion requests per day and can support peaks of more than 20 million requests per second.

In this project, DynamoDB is used to store the major events that occur in the system, for log purposes.

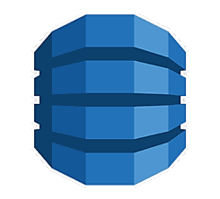


Figure 13 - AWS DynamoDB Logo

### Cognito

AWS Cognito is an Amazon Web Services (AWS) product that controls user authentication and access for mobile applications on internet-connected devices. The service saves and synchronizes end-user data, which enables an application developer to focus on writing code instead of building and managing the back-end infrastructure. This can accelerate the mobile application development process.

In this project, AWS Cognito service is used as a sign-in method for the application user.

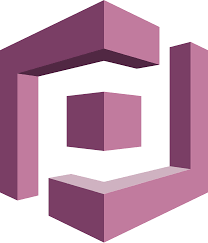


Figure 14 - AWS Cognito Logo

### IAM

AWS Identity and Access Management (IAM) is a web service that helps you securely control access to AWS resources. You use IAM to control who is authenticated (signed in) and authorized (has permissions) to use resources.

When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account root user and is accessed by signing in with the email address and password that you used to create the account.

In this project, AWS IAM service is used to secure the AWS resources and give permissions only to specific components in the system, for specific operations.



Figure 15 - AWS IAM Logo

### ACM

AWS Certificate Manager is a service that lets you easily provision, manage, and deploy public and private Secure Sockets Layer/Transport Layer Security (SSL/TLS) certificates for use with AWS services and your internal connected resources. SSL/TLS certificates are used to secure network communications and establish the identity of websites over the Internet as well as resources on private networks. AWS Certificate Manager removes the time-consuming manual process of purchasing, uploading, and renewing SSL/TLS certificates.

With AWS Certificate Manager, you can quickly request a certificate, deploy it on ACM-integrated AWS resources, such as Elastic Load Balancers, Amazon CloudFront distributions, and APIs on API Gateway, and let AWS Certificate Manager handle certificate renewals.

In this project, AWS ACM service is used to provide a certificate for the external domain used by the video streaming feature.

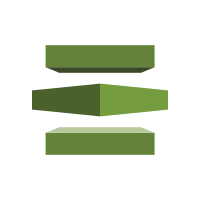


Figure 16 - AWS ACM Logo

# Additional Software Libraries

## OpenCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products.

In this project, OpenCV is used to perform preliminary face detection on the images captured by the Raspberry Pi camera, before sending the image to Rekognition. This process reduces the number of false positives, which consume both time and bandwidth.

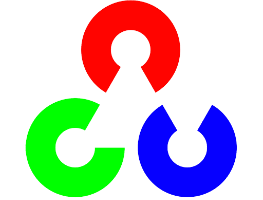


Figure 17 - OpenCV Logo

## UV4L

Userspace Video for Linux (UV4L) is a service that handles the video stream as a device file (/dev/video0).

UV4L contains many other services support (including Jitsi) and can use webRTC with those services.

WebRTC ("Web Real-Time Communication") is a collection of communications protocols and application programming interfaces that enable real-time communication over peer-to-peer connections.

## Jitsi

Jitsi is a set of open-source projects that allows you to easily build and deploy secure videoconferencing solutions.

The main project is Jitsi Meet – Secure, Simple and Scalable Video Conferences that you use as a standalone app or embed in your web application.

Jitsi Meet includes many features and we are constantly adding more. Some of the main ones are:

* Auto-view the active speaker or click on any attendee to see their video
* Android and iOS apps
* Text chatting (web only)
* Lock a room with a password
* Screen sharing (if jidesha is setup, only required in Chrome)
* Streaming a conference to YouTube live (if Jibri is configured)
* Shared text document based on Etherpad
* Raise/Lower your hand for attention
* Participant talk-time statistics
* Push-to-talk mode
* Play a YouTube video to all attendees’ call
* Audio-only option
* Telephone dial-in to a conference (if Jigasi is setup)
* Dial-out to a telephone participant (if Jigasi is setup)
* **Integration in other apps / websites**

In this project Jitsi Meet is used, mainly its last **highlighted** feature, alongside UV4L, to stream the video from the Raspberry pi to the EC2 instance for the application users to view.



Figure 18 - Jitsi Logo

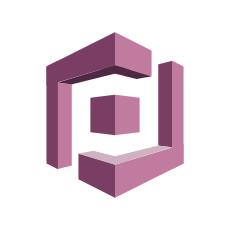
# System Architecture



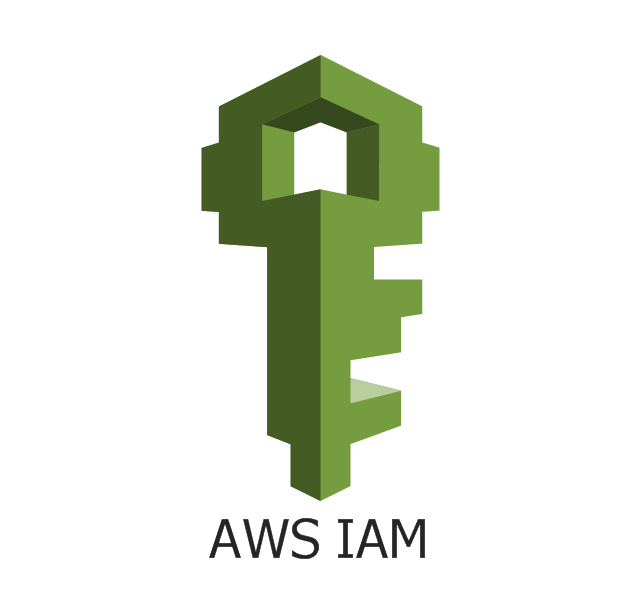
NoSQL

MQTT

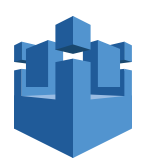
+ FCM



AWS Cognito



Authentication



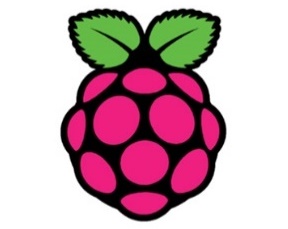
Amazon IoT



**UV4L**



Google FCM



Raspberry Pi + Camera Module

MQTT

+ FCM

NoSQL

+ Face recognition

Video Streaming

Video Streaming

Sign-in



Figure 19 - System Architecture

Amazon

DynamoDB

The Raspberry Pi continuously captures and analyses images from its camera module.  
When a recognized person is detected (using OpenCV and AWS Rekognition), the RP unlocks door for a constant duration to allow the person entry, and afterward it locks the door.

When an unrecognized person presses the “ring” button, the RP notifies the owner via the Android application (using Google FCM) with a snapshot of the unrecognized person.

The owner, using the Android app, can then perform the following actions:

1. Unlock the door to allow entry to the unrecognized person (the door locks back automatically after a constant duration) (using AWS IoT)
2. Add the unrecognized person to the recognized persons database (for the specific door) (using AWS IoT and DynamoDB)
3. View a live stream (using AWS EC2, UV4L and Jitsi) of the door

Note: Actions (1) and (3) can be performed at any time.

Additionally, the owner can view a log of activities of his/her system.

# Raspberry Pi Detailed Design

<https://github.com/aamnony/Smart-Doorbell/blob/master/raspberry_pi/__main__.py>

The RP runs a Python script which consists of a finite state machine (FSM) running indefinitely on a thread, with two supporting threads for MQTT and ring button listeners which update two variables which the FSM reads):

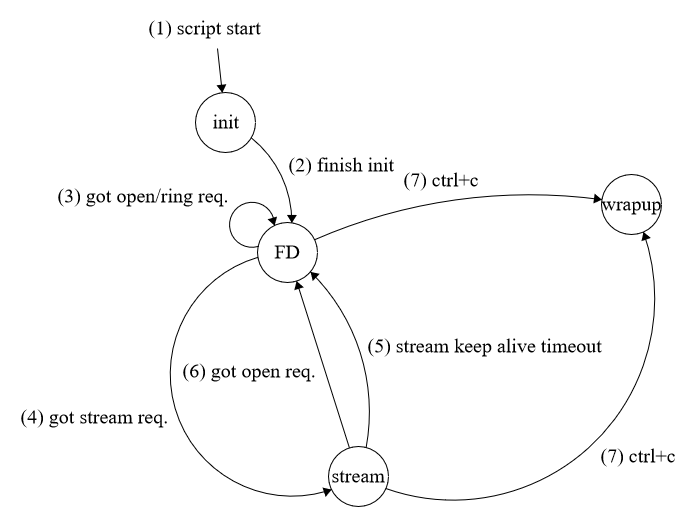


Figure 20 - Raspberry Pi Finite State Machine

1. Upon starting the script, the FSM is created and initialized

**State – init:**

Initialize video capturing.

Create MQTT, FCM and ring button clients and listeners.

1. Upon finishing initialization, the FSM starts face detecting (FD)

**State – FD (face detecting):**

Capture a still image and preprocess it using OpenCV:

* 1. Test if the image contains a face
  2. If the last FRAMES\_WITH\_FACES\_THRESHOLD consecutive images contained faces – upload the last image to AWS S3 (for log purposes) and run it through AWS Rekognition. If AWS Rekognition recognized the face – unlock the door DEFAULT\_UNLOCK\_DURATION seconds and log the unlock action in AWS DynamoDB.

1. When receiving the following requests:
   1. If received an open request from Android app – unlock the door for DEFAULT\_UNLOCK\_DURATION seconds.
   2. If received a ring request from the ring button –
      1. Capture and upload the image to AWS S3 (for log purposes)
      2. Send a notification using FCM to Android app with the captured image
      3. Log the ring action in AWS DynamoDB
2. If received a stream request from Android app – go to stream state

**State – stream:**Stop the video capturing (Jitsi requires control over the camera module).

Open a chromium browser with the Jitsi room (ROOM\_URL).

1. No stream request (i.e. keep alive signal) was received for STREAM\_KEEP\_ALIVE\_TIMEOUT seconds –
   1. Close the chromium browser
   2. Start the video capturing
   3. Go to FD state
2. If received an open request from Android app – unlock the door for DEFAULT\_UNLOCK\_DURATION seconds and go to FD state.
3. Upon receiving keyboard interrupt (Ctrl+C) – wrap-up and close all resources.

## Required External Packages

AWSIoTPythonSDK

boto3

cv2

firebase\_admin

gi

Wnck

pynput

RPi.GPIO

## Electrical Circuit

The RP circuit consists of a pushbutton which act as a ringer, and a lock, as described in the following diagram:

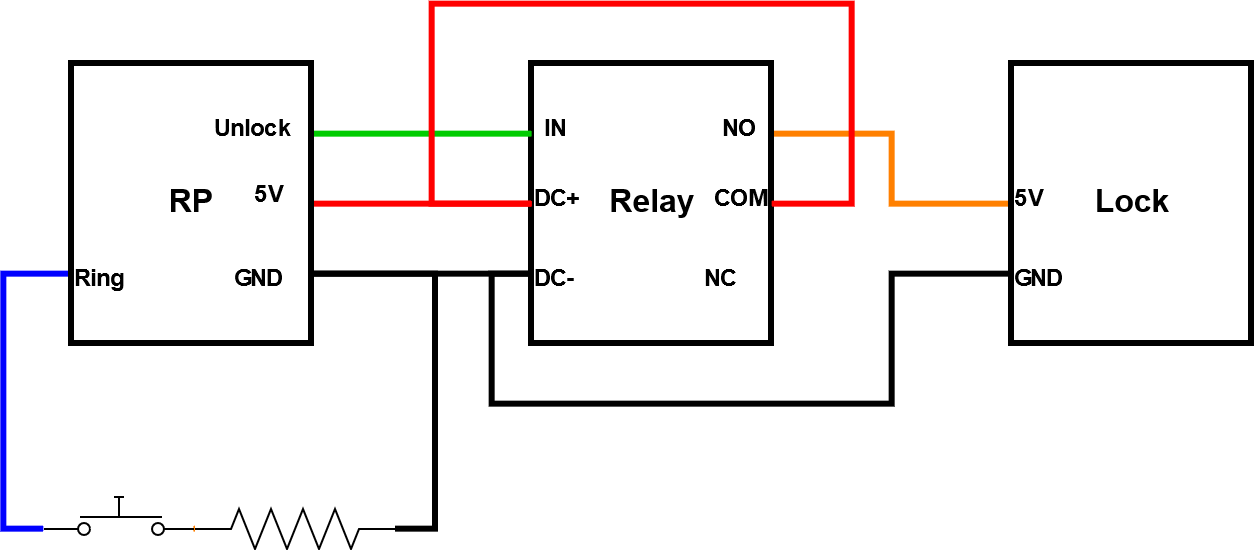


Figure 21 - RP Electrical Circuit

The relay is used to drive the lock due to lack of current from the PI user pins (i.e. the “Unlock” pin in the diagram).

# EC2 Detailed Design

The EC2 instance runs UV4L and Jitsi on Ubuntu 18.04 to provide the video streaming capabilities.

UV4L and Jitsi were installed by following the official UV4L guide ‎(17).

The EC2 instance is hosted on the AWS domain but is forwarded from the URL: <https://www.smartdoorbell.ga/> (provided by [www.freenom.com](http://www.freenom.com)) which we certified using AWS ACM.

# Android Detailed Design

<https://github.com/aamnony/Smart-Doorbell/tree/master/android>

The Android app was developed in Android Studio environment and contains activities and supporting classes.

The following diagram describes the main activities and classes of the app:

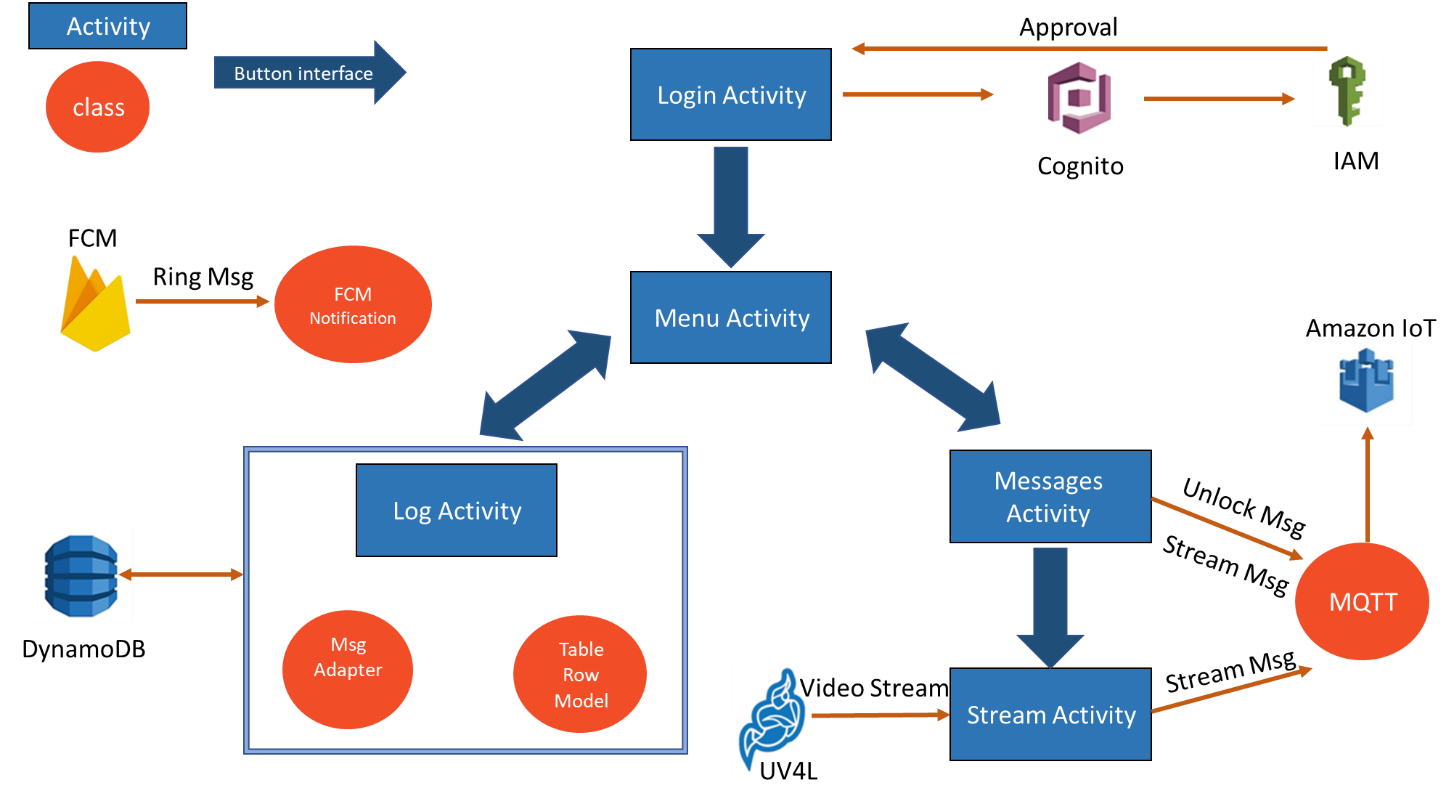


Figure 22 - App Overview

The following sections describe the main activities and classes of the app.

## LoginActivity

<https://github.com/aamnony/Smart-Doorbell/blob/master/android/app/src/main/java/com/github/aamnony/smartdoorbell/LoginActivity.java>

<https://github.com/aamnony/Smart-Doorbell/blob/master/android/app/src/main/res/layout/activity_login.xml>

This activity allows the user to login with the home's username as password.  
The registration of a user to the system is done manually on AWS configuration.   
The activity is based on a typical login design supplied by AWS, based on instance of AWSMobileClient type.  
In case of success (the username and password approved by IAM through Cognito), the app switches to MenuActivity. Otherwise, an error message is displayed.  
The login operation also creates a FireBase instance (which configures FCM by itself) and initializes it to work.

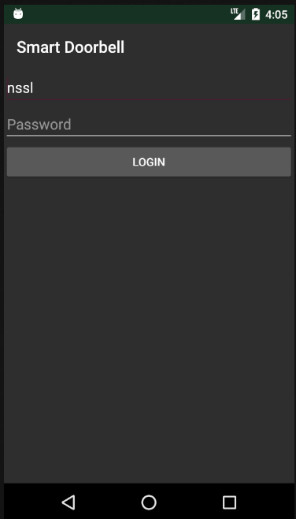


Figure 23 – LoginActivity

## MenuActivity

<https://github.com/aamnony/Smart-Doorbell/blob/master/android/app/src/main/java/com/github/aamnony/smartdoorbell/MenuActivity.java>

<https://github.com/aamnony/Smart-Doorbell/blob/master/android/app/src/main/res/layout/activity_menu.xml>

This activity acts as the gate to all other screens of the application. It's a basic activity that has buttons leading to the Log screen and to the Actions screen.

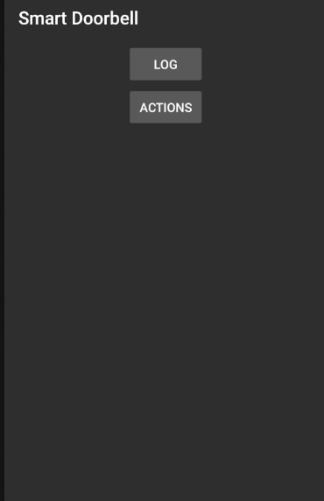


Figure 24 - MenuActivity

## LogActivity

<https://github.com/aamnony/Smart-Doorbell/blob/master/android/app/src/main/java/com/github/aamnony/smartdoorbell/LogActivity.java>

<https://github.com/aamnony/Smart-Doorbell/blob/master/android/app/src/main/res/layout/activity_log.xml>

The system log is saved to a DynamoDB table.  
This activity uses DynamoDBMapper to pull the items from the table and displays them in a ListView using MsgAdapter.

### MsgAdapter and TableRowModel

<https://github.com/aamnony/Smart-Doorbell/blob/master/android/app/src/main/java/com/github/aamnony/smartdoorbell/MsgAdapter.java>

<https://github.com/aamnony/Smart-Doorbell/blob/master/android/app/src/main/java/com/github/aamnony/smartdoorbell/TableRowModel.java>

This class is used to display the log items in a ListView.   
This class derives from the BaseAdapter class, and uses the model defined in TableRowModel as its table row model which defines the query that is made on the database (including the attribute names and methods to pull each field).  
Each row in the log ListView contains the following fields:

* Date of the action
* Name of recognized person, if exists
* Action type
* Camera snapshot during the action

Some actions don't have snapshots or person names (e.g. stream action), so default values are displayed ("null" for the person name, empty image for the snapshot).

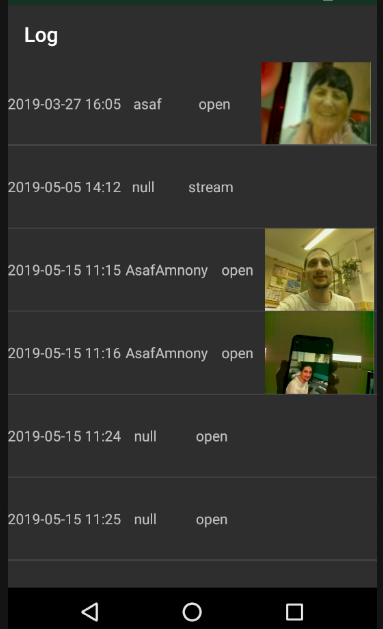


Figure 25 - LogActivity

## MessagesActivity

<https://github.com/aamnony/Smart-Doorbell/blob/master/android/app/src/main/java/com/github/aamnony/smartdoorbell/MessagesActivity.java>

<https://github.com/aamnony/Smart-Doorbell/blob/master/android/app/src/main/res/layout/activity_messages.xml>

This activity allows the user to perform two main actions:

* Unlock the door
* View stream

These actions are performed by the MQTT class.

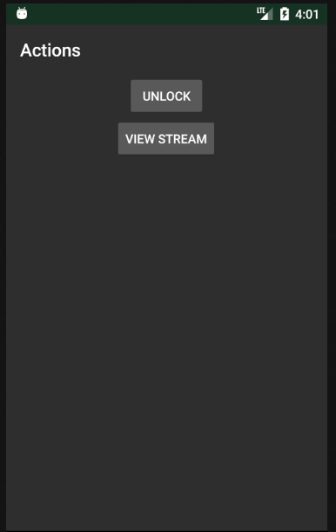


Figure 26 - MessagesActivity

## MQTT

<https://github.com/aamnony/Smart-Doorbell/blob/master/android/app/src/main/java/com/github/aamnony/smartdoorbell/Mqtt.java>

This class sends messages to the RP using MQTT protocol (messages from RP to Android are sent by FCM).

The following messages are supported:

|  |  |
| --- | --- |
| Action | Message Format |
| Stream | stream |
| Unlock | open |
| Unlock & Add | open <added\_person\_name> <snapshot\_id> |

## StreamActivity

<https://github.com/aamnony/Smart-Doorbell/blob/master/android/app/src/main/java/com/github/aamnony/smartdoorbell/StreamActivity.java>

This activity is responsible for the video streaming. It extends JitsiMeetActivity that is supplied directly from Jitsi. Two extensions were added:

* An option of unlocking the door directly from this activity.
* A messages mechanism that sends keepalive messages to the RP:

While the activity is alive, it sends MQTT "stream" messages, so that the RP keeps streaming video. After a timeout without these messages, the RP stops streaming.



Figure - StreamActivity

## FcmNotification

<https://github.com/aamnony/Smart-Doorbell/blob/master/android/app/src/main/java/com/github/aamnony/smartdoorbell/FcmNotification.java>

<https://github.com/aamnony/Smart-Doorbell/blob/master/android/app/src/main/java/com/github/aamnony/smartdoorbell/FcmService.java>

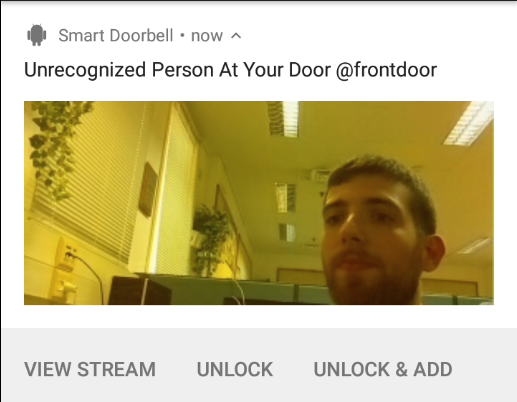
This class parses “Ring” messages received from the RP via FCM service and displays them as Android notifications.

Each message contains a Map<String, String> containing the following data:

|  |  |
| --- | --- |
| Key | Value |
| camera\_name | The name/ID of the camera/RP that sent the message (the username is taken from the currently logged user) |
| image\_name | The name of the image snapshot in the S3 bucket, at the time when the doorbell was rung |

Each notification contains:

1. A snapshot the camera took when the doorbell was rung.
2. Button to view stream.
3. Button to unlock the door.
4. Button to unlock the door and add the current snapshot to the recognized faces database.



**1**

**2**

**3**

**4**

Figure 28 - FcmNotification

# Issues and Resolutions

|  |  |
| --- | --- |
| Issue | Resolution |
| Failed installation of U4Vl and Jitsi on the EC2 machine | Switched to an Ubuntu 18.04 machine |
| Certificate issues for the Jitsi server | Used an external domain, connected it to our ELB DNS, which is connected to our EC2 instance. Got an AWS ACM certificate for the external domain |
| AWS Rekognition on Raspberry Pi - exporting environment variables didn’t work | Needed to add the exports to script (aws.sh) inside /etc/profile.d/ |
| Physical lock couldn’t be toggled directly from PI outputs because of lack of current, but was able using the PI power (which is driven directly from wall adapter) | Used relay |
| Camera module not supported | Installed TC358743 driver |
| Camera module supported but not detected | Replaced camera module flat cable connector |
| Camera snapshot using UV4L didn't work | Updated PI firmware: rpi-update.  Also installed ALL packages above line: "for which the manual pages are available" in the UV4L install guide |
| OpenCV installation failed | Changed default Python interpreter to Python 2 even though we used Python 3 for running OpenCV.  After installation finished, we returned to Python 3 |
| OpenCV installation - failed countless times, including freezing and ruining the SD-card | Used 1 job instead of 4 jobs:  make -j1 (instead of -j4) |
| DynamoDB let a role access table for a few hours after the table creation even if the role has no permission |  |
| Jitsi released a new version (in which the API is backward incompatible) and Android compiled the new version because of implicit gradle library version ("+") | Used explicit gradle library versions for all libraries in Android app |

# Future Tasks

In its current state the system supports a single user and a single camera:

* New users need to be registered in AWS manually by the system administrator
* The Android app supports a specific camera in streaming
* The RP is designed as a single camera for a specific user and can be configured easily as a different camera for a different user
* We designed all AWS infrastructure, so it supports multi-users and multi-cameras without further adjustments

Some tasks for the future:

* Create an automatic registration process
* Add support for multi-cameras in Android app
* Secure MQTT communication

# Bibliography

1. OpenCV install guide - <https://www.pyimagesearch.com/2017/09/04/raspbian-stretch-install-opencv-3-python-on-your-raspberry-pi/>
2. Issue and solution when installing OpenCV - <https://lb.raspberrypi.org/forums/viewtopic.php?p=1291404&sid=b82b69c683ca32bec94e8520bd965df0#p1291404>
3. Checking if camera module is connected - <https://www.raspberrypi.org/forums/viewtopic.php?t=174375>
4. Setup camera module - <https://www.raspberrypi.org/documentation/configuration/camera.md>
5. Camera module Python guide - <https://projects.raspberrypi.org/en/projects/getting-started-with-picamera>
6. Raspberry Pi GPIO usage 1 - <https://learn.sparkfun.com/tutorials/raspberry-gpio/python-rpigpio-api>
7. Raspberry Pi GPIO usage 2 - <https://www.raspberrypi.org/documentation/usage/gpio/>
8. Raspberry Pi GPIO events - https://stackoverflow.com/questions/16143842/raspberry-pi-gpio-events-in-python
9. Relay usage - <http://www.circuitbasics.com/setting-up-a-5v-relay-on-the-arduino/>
10. Raspberry Pi install guide - <https://www.raspberrypi.org/documentation/installation/installing-images/>
11. Changing Python default interpreter version - <https://linuxconfig.org/how-to-change-from-default-to-alternative-python-version-on-debian-linux>
12. SD-Card backup guide - <https://thepihut.com/blogs/raspberry-pi-tutorials/17789160-backing-up-and-restoring-your-raspberry-pis-sd-card>
13. AWS SDK Android samples - <https://github.com/awslabs/aws-sdk-android-samples>
14. AWS Cognito backend intro - <https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_providers_oidc_cognito.html>
15. AWS Cognito frontend intro - <https://docs.aws.amazon.com/aws-mobile/latest/developerguide/getting-started.html>
16. AWS Cognito frontend guide - <https://docs.aws.amazon.com/aws-mobile/latest/developerguide/add-aws-mobile-user-sign-in.html>
17. Jitsi install guide - <https://www.linux-projects.org/uv4l/tutorials/jitsi-meet/>