California State Polytechnic University, Pomona
Department of Electrical and Computer Engineering

SASBOT
Microcontroller Based Smart Access System with Slack Bot Assistant

Senior Design Project

By: Ashot Hambardzumyan
and
Kaywan Balbas

Advisor: Meng-Lai Yin

Table of Contents

Objective:

When designing the **S**mart **A**ccess **S**ystem with Slack **Bot** Assistant (SASBOT), the objective was to modernize access systems for buildings and communities. Instead of using archaic technologies, such as land line phones and mechanical keypad interfaces, SASBOT facilitates communication through an intuitive touch screen connected to Wi-Fi. SASBOT provides a simple and intuitive interface that facilitates an efficient communication experience for both guests and tenants.

Project Summary

The project went through several revisions. We started off by sending text messages to cell phones from a microcontroller, but very soon we realized the limitations involved with text messages. Some of these limitations included: a very long delay in receiving the text, some messages would never be received, and there was an absence of any sort of authorization for who could grant access on behalf of a specific tenant.

To compromise the above issues, Twilio and AWS were considered. Twilio allows phone calls over the internet – also referred to as Voice over IP (VoIP) – and AWS allows tracking device status and sending messages (SMS) to cell phones. Twilio and AWS provide the libraries necessary to implement VoIP and SMS on a microprocessor but not on a microcontroller. Switching to a microprocessor like the Raspberry Pi could have solved that issue, but it would have made the project about twice as expensive and it seemed like an easy shortcut.

Finally, we discovered Slack. Slack was the right balance between cost, performance, and complexity. Using Slack allowed us to implement push notifications, SMS messaging and issuing commands from a smartphone, while still using a very simple and inexpensive microcontroller. In this report we have included detailed information describing the design, development, and testing processes for this project.

Through the completion of this project, we were able to expand our knowledge as Computer Engineers significantly. This project allowed us to apply past experiences obtained from previous courses and build upon them with new technologies that aren't taught inside the classroom. We also had the opportunity to test our own creativity and our ability to take an idea, turn it into a concept, then build an actual product. From the start, our goal was to learn; despite the many obstacles we faced along the way, this project proved to be an invaluable learning experience.

Glossary

IoT:
The Internet of Things (IoT) refers to the ever-growing network of physical objects that feature an IP address for internet connectivity, and the communication that occurs between these objects and other Internet-enabled devices and systems.

Slack:
Slack is a group messaging application full of many useful features. Slack is available on any device, including desktop, iOS, and Android.

Bot:
In the context of our project, a bot is a programmed "robot" user, implemented within the Slack application. The bot has the ability to send and receive messages, as well as access other features available on Slack.

AWS Lambda:
Amazon Web Services (AWS) Lambda lets you run code without provisioning or managing servers. You pay only for the compute time you consume - there is no charge when your code is not running. With Lambda, you can run code for virtually any type of application or backend service - all with zero administration. Just upload your code and Lambda takes care of everything required to run and scale your code with high availability. You can set up your code to automatically trigger from other AWS services or call it directly from any web or mobile app.

MCU:
A microcontroller unit (MCU) is a small computer on a single integrated circuit, designed for embedded applications.

Amazon API Gateway:
Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale.

FTDI:
FTDI is a commonly used USB to UART chip. Refer to materials list for more information.

ESP12:
ESP12 is a popular, inexpensive MCU with built-in Wi-Fi. It was initially released as a Wi-Fi module. However, given its processing power, the IoT community turned it into fully functional MCU. Refer to materials list for more information.

Nextion Display:
Nextion is a new touchscreen display series from ITEAD. These displays have dedicated graphic processors. Refer to materials list for more information.

Concept of Operations

Figure 1 shows a high level description of the system operations. The system is initiated when a guest requests access to the building. The guest uses the SASBOT Display to find the profile page of the tenant they are visiting. The guest will then press the message icon, and the SASBOT will notify the selected tenant. The tenant will then receive a push notification on their smartphone from the Slack app. Upon opening the app, the tenant can use one of two buttons in their Slack feed to either grant or deny access. The SASBOT Display will show a message to notify the guest if access is granted or denied. If the tenant doesn't respond within 20 seconds, the guest will receive a message on the display notifying them that the tenant has not responded. At this point, the request for access will expire, and the guest must send another request if they still wish to gain access. This is the basic flow of the system. However, a guest can choose to use the keypad on the SASBOT display to enter a 4-digit passcode that will either grant or deny access.

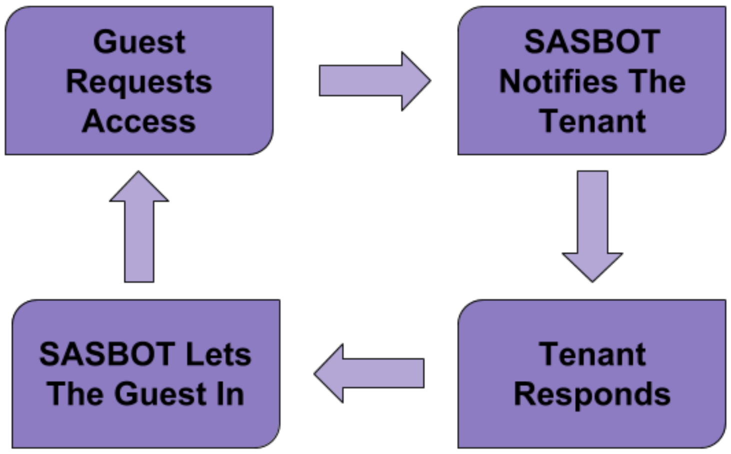


Figure 1. Concept of Operations

Block Diagram

SASBOT is a distributed system. No one core system runs it all. A system-level block diagram is shown in Figure 2. Each component has some autonomous functionality. When the guest requests access using the screen, information is sent to the MCU which is then sent to the tenant's smartphone. Now, the system waits for the tenant's response. When the tenant responds, the information is forwarded to the MCU then to the screen to be displayed.

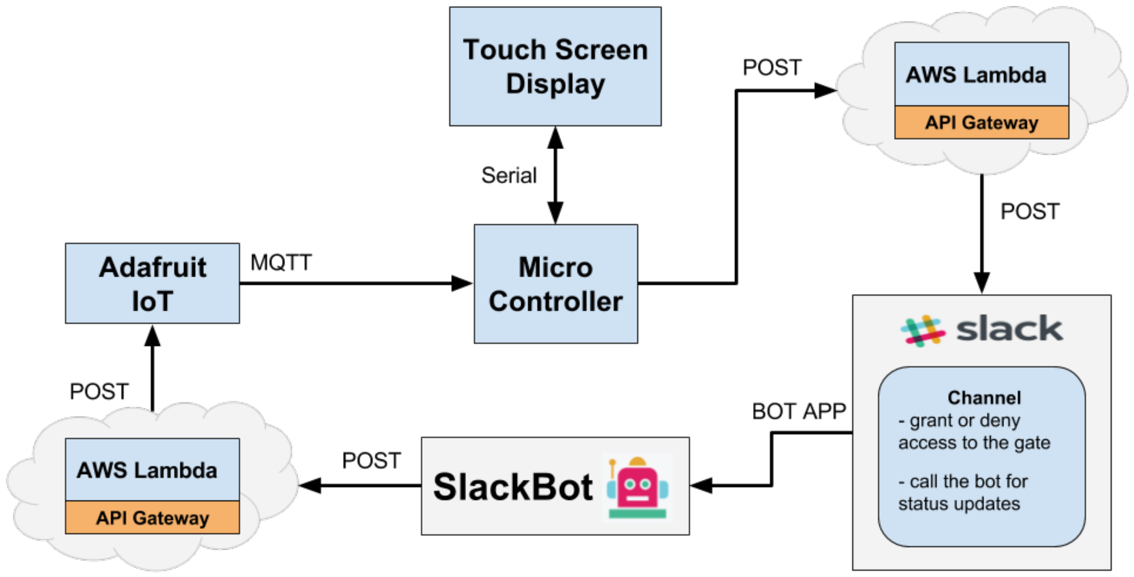


Figure 2. System-Level Block Diagram

The display contains all the graphical elements and does the necessary processing. It can navigate between pages and show graphics without the MCU. The MCU is the arbiter that controls and coordinates different components. It triggers an AWS Lambda function when a guest requests access and displays the tenant's response.

To simplify MCU to Lambda communication, API Gateway is used. API Gateway uses simple HTTP POST method, contrary to HTTPS with custom authentication protocol used by Lambda. AWS Lambda is responsible for posting to Slack and listening for the tenant's response.

Slack facilitates communication between residents, and it enables rich developer functionalities through SlackBot. With SlackBot, we can post prompts to the residents and take actions based on their interactions. If the tenant responds to the prompt, Slackbot will trigger another AWS Lambda to post on Adafruit IOT. Adafruit IOT sends this information to the MCU which in turn signals the display to show the appropriate message.

Wiring Diagrams

Follow Figure 3 to connect the ESP12 to a computer using a USB connection. The ESP12 does not have a USB connection, so we used an FTDI module to flash new software onto it. After loading the software, disconnect the ESP12 and connect to the screen according to Figure 4. Follow the Table 1 to put the ESP12 into a programming mode after connecting according to Figure 3.

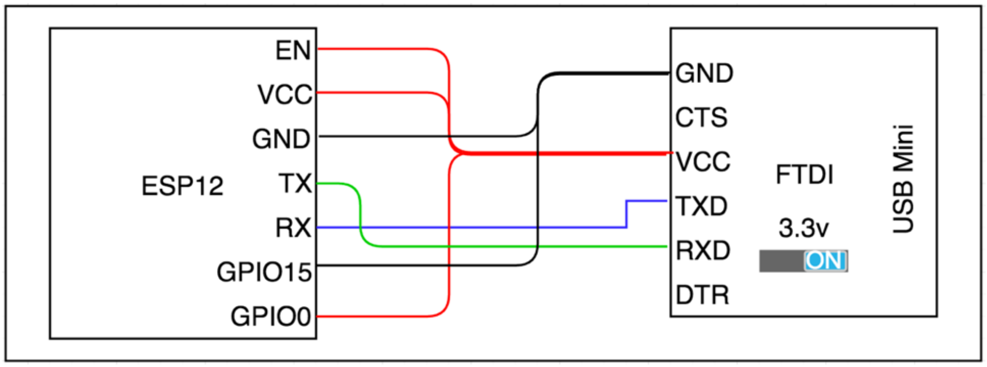


Figure 3. FTDI to ESP12

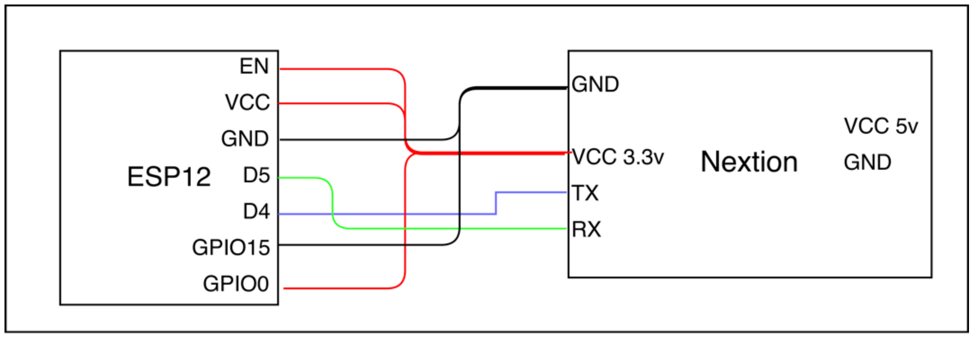


Figure 4. ESP12 to Nextion

Programming ESP12

In order to load a new program onto the ESP12, it needs to be put into programming mode first. The MCU is programmed using a UART connection. We used a USB to UART module (FTDI) to connect the MCU to the computer. For the programming environment, we used the Eclipse IDE with Eclipse C++ IDE for Arduino plugin. This plugin allows access to open source libraries, compiles to selected board and includes a serial terminal for loading a new program and debugging.

We started off with a simple ESP12 and FTDI, but later on decided to use the NodeMCU. The NodeMCU is a development board with the ESP12 and FTDI built onto the same board. The NodeMCU board is a little larger in size, but it simplifies wiring and programming. When using the NodeMCU, wiring the ESP12 to FTDI and putting the ESP12 into programming mode can be skipped.

Table 1. Putting ESP12 in Programming mode

Steps to put ESP12 into programming mode	
Step 1	Set Serial connection to 115200baud
Step 2	Connect GPIO0 to ground
Step 3	Restart the ESP12
Step 4	Upload the sketch
Step 5	Connect GPIO0 to VCC (3.3v)
Step 6	Restart the ESP12
Step 7	Toggle EN (High-Low-High)

3D-Printed Enclosure

After connecting the microcontroller and the screen together, we needed an enclosure to house it all. We wanted something with a low profile. Fortunately, we were able to find this enclosure online. This enclosure was downloaded from an open source community for 3D models. It is a simple design with holes for wires, screws, and a standard tripod mount. We printed this enclosure using a Thinkine 3D printer. Using this enclosure gave our project a more complete and professional look.

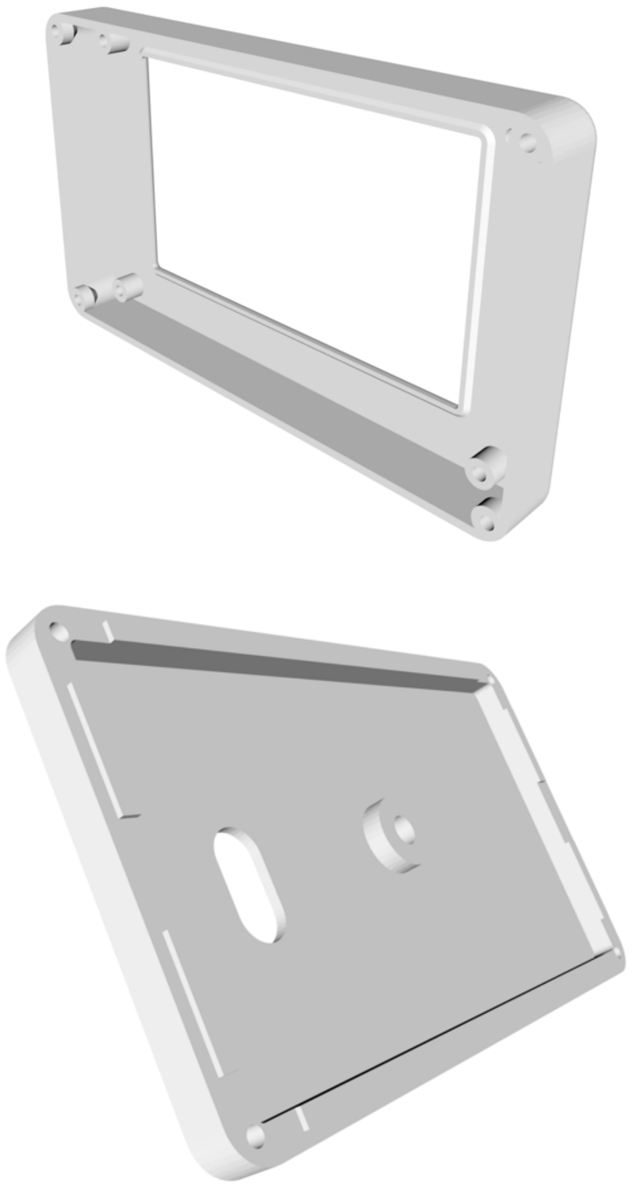


Figure 5. 3D Enclosure

Materials

We kept the project relatively low budget, so it doesn't require much in terms of hardware. In addition, the web services used were at no cost since AWS, Slack and Adafruit offer free, limited service.

Initially we purchased the ESP12 and an FTDI module separately, but we later realized that buying a development board that combines these two modules into one simplifies the wiring and programming. This development board is called NodeMCU. It is slightly bigger so it did not fit into the enclosure. However, it is a good option during the development phase. Refer to Table 2 for material list.

The touch screen display is the Nextion HMI Touchscreen Display, manufactured by Itead. The model we used is 4.3 inches diagonally.

Table 2. Materials List

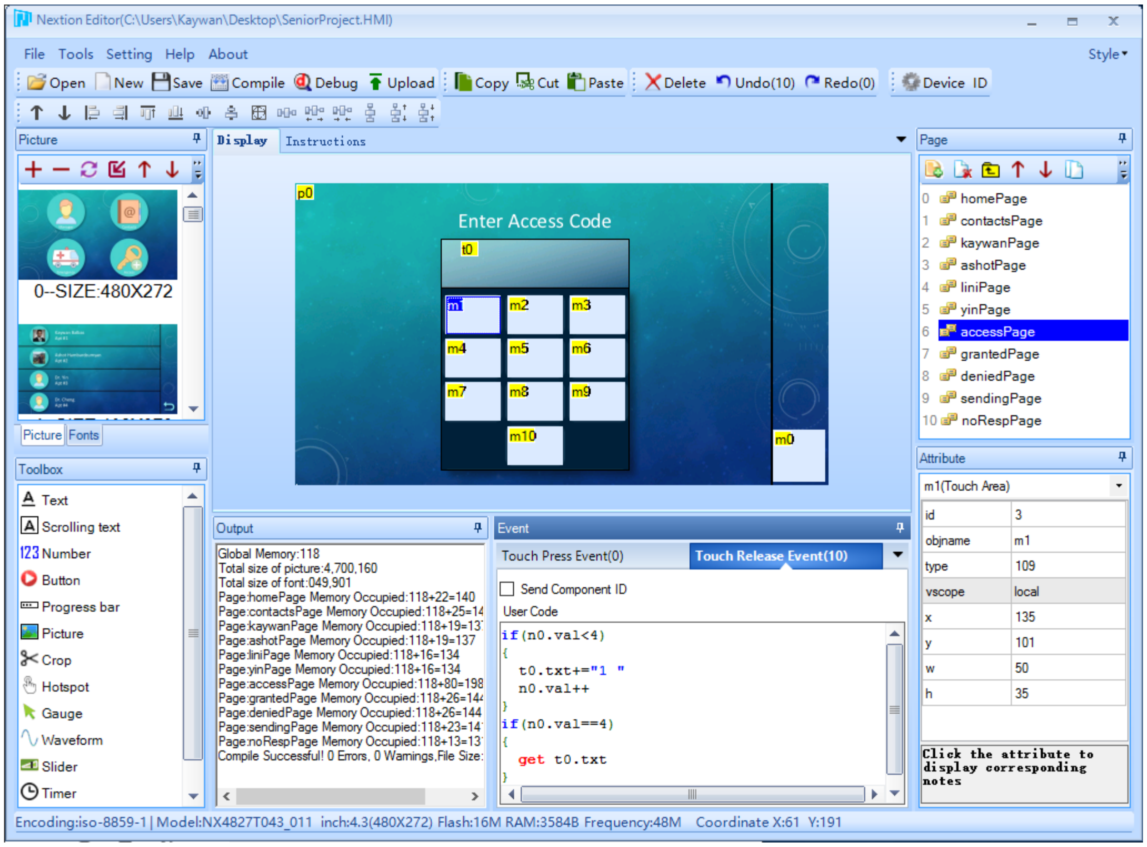
1. Display	Itead Nextion HMI Touchscreen Display (NX4827T043)
2. Microcontroller	ESP12
3. USB to UART	Micro USB FTDI
4. MCU w/USBtoUart	NodeMCU

Nextion Editor IDE

Figure 6 below is a screenshot of the Nextion Editor IDE used to program the user interface for the SASBOT. Nextion Editor is specifically designed to be used with Nextion displays. It has many features which make it convenient and fairly easy to use.

One great feature is the ability to import a PNG image and overlay it with touch area boxes to program different actions when a certain area on the screen is pressed. This means the graphics can be developed in a separate application, such as PowerPoint. In general, this is much easier than any type of GUI programming in languages such as C++ or Java. Using this feature allowed us to focus more on the functionality of the SASBOT, rather than our ability to program a nice user interface. All of the “pages”, or “views” as some may call it, were first designed within PowerPoint, the imported into Nextion Editor. Within Nextion Editor, this is still some programming required. Any sort of touch press or release event is programmed within Nextion Editor to either change to a different screen, or to send some sort of data to the microcontroller.

Another useful feature is the ability to run a simulation within the IDE before uploading it to the actual display. The simulation is accessed by using the Debug button. This was a very powerful tool for testing because it allowed us to quickly test functionality of certain components of the interface. It also helped us learn the format in which data was being sent from the screen to the microcontroller.



```
Global Memory:118
Total size of picture:4,700,160
Total size of font:049,901
Page:homePage Memory Occupied:118+22=140
Page:contactsPage Memory Occupied:118+25=14
Page:kaywanPage Memory Occupied:118+19=13
Page:ashotPage Memory Occupied:118+19=137
Page:liniPage Memory Occupied:118+16=134
Page:yinPage Memory Occupied:118+16=134
Page:accessPage Memory Occupied:118+80=198
Page:grantedPage Memory Occupied:118+26=14
Page:deniedPage Memory Occupied:118+26=144
Page:sendingPage Memory Occupied:118+23=14
Page:noRespPage Memory Occupied:118+13=13
Compile Successful! 0 Errors, 0 Warnings,File Size:
```

```
if(n0.val<4)
{
  t0.txt+="1 "
  n0.val++
}
if(n0.val==4)
{
  get t0.txt
}
```

id	3
objname	m1
type	109
vscope	local
x	135
y	101
w	50
h	35

Click the attribute to display corresponding notes

Encoding:iso-8859-1 | Model:NX4827T043_011 inch:4.3(480X272) Flash:16M RAM:3584B Frequency:48M Coordinate X:61 Y:191

Figure 6. Nextion Editor IDE

Appendix A
Software Code

Code for the MCU
(4 files in C++)

proj_globals.h

```
/*
 * proj_globals.h
 *
 *  Created on: Apr 10, 2017
 *      Author: Ash
 */

#ifndef PROJ_GLOBALS_H_
#define PROJ_GLOBALS_H_

#include <Arduino.h>
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include "Adafruit_MQTT.h"
#include "Adafruit_MQTT_Client.h"
#include "Nextion.h"
#include <SoftwareSerial.h>
#include <String.h>

#include <DNSServer.h>          //Local DNS Server used for redirecting all requests to the configuration portal
#include <ESP8266WebServer.h>   //Local WebServer used to serve the configuration portal
#include <WiFiManager.h>        //https://github.com/tzapu/WiFiManager WiFi Configuration Magic


/************************* Adafruit.io Setup *********************************/

#define AIO_SERVER      "io.adafruit.com"
#define AIO_SERVERPORT  1883                   // use 8883 for SSL
#define AIO_USERNAME    "Your Username Here"
#define AIO_KEY         "Your Key Here"

/************************* Global State  ********************************/

// Create an ESP8266 WiFiClient class to connect to the MQTT server.
WiFiClient client;

// Setup the MQTT client class by passing in the WiFi client and MQTT server and login details.
Adafruit_MQTT_Client mqtt(&client, AIO_SERVER, AIO_SERVERPORT, AIO_USERNAME, AIO_KEY);

/****************************** Feeds ***************************************/

// Setup a feed called 'reply' for subscribing to changes.
```

```
Adafruit_MQTT_Subscribe accessFeed = Adafruit_MQTT_Subscribe(&mqtt, AIO_USERNAME
"/feeds/access");
Adafruit_MQTT_Subscribe postupdateApt1Feed = Adafruit_MQTT_Subscribe(&mqtt, AIO_USERNAME
"/feeds/postupdateapt1");
Adafruit_MQTT_Subscribe postupdateApt2Feed = Adafruit_MQTT_Subscribe(&mqtt, AIO_USERNAME
"/feeds/postupdateapt2");
Adafruit_MQTT_Subscribe postupdateManagerFeed =
			Adafruit_MQTT_Subscribe(&mqtt, AIO_USERNAME "/feeds/postupdatemanager");


/************************* Sketch Code ***********************************/
#define PIN "1 2 3 4 "
#define DENY "page deniedPage"
#define GRANT "page grantedPage"
#define PINPAGE 6
#define SENDINGPAGE 9

SoftwareSerial nextionSerial(D4, D5); // RX, TX
Nextion myNextion(nextionSerial, 9600);
WiFiClientSecure secureClient;
//char* apt; //todo set to users apt
String text = ""; //null pointer
String url = "/prod/PostToSlack";
String access = ""; //null pointer
String updateMess = ""; //null pointer
uint8_t pageID = 0; //init to 0 unsigned char
const char* host = "Your AWS API Gateway for Slack Post";
const int httpsPort = 443;
const char* fingerprint = "80 C7 E4 AF 9B DB 3D 9A 09 25 28 8A EF 3B 1C 9B 08 47 70 A4";
Adafruit_MQTT_Subscribe *subscription;
WiFiManager wifiManager;

#endif /* PROJ_GLOBALS_H_ */
```

proj_setup.h

```
/*
 * proj_setup.h
 *
 *  Created on: Apr 10, 2017
 *      Author: Ash
 */
#include "proj_globals.h"

#ifndef PROJ_SETUP_H_
#define PROJ_SETUP_H_

void setup() {
	//setup HW serial
	Serial.begin(115200);

	//to configure WIFI for the connect to SASBOT with
```

```
            //password of "password" and follow the prompts
            //to reset WIFI go to pin page and enter 1993,
            //then restart the system.
            wifiManager.autoConnect("SASBOT", "password");
            Serial.println("WiFi connected");
            Serial.println("IP address: ");
            Serial.println(WiFi.localIP());

            //Nextion
            myNextion.init();

            // Setup MQTT subscriptions
             mqtt.subscribe(&accessFeed);
             mqtt.subscribe(&postupdateApt1Feed);
             mqtt.subscribe(&postupdateApt2Feed);
             mqtt.subscribe(&postupdateManagerFeed);
}
#endif /* PROJ_SETUP_H_ */
```

proj_funcs.h

```
/*
 * proj_funcs.h
 *
 *  Created on: Apr 10, 2017
 *      Author: Ash
 */
#include "proj_globals.h"


#ifndef PROJ_FUNCS_H_
#define PROJ_FUNCS_H_

void MQTT_connect() {
  int8_t ret;

  // Stop if already connected.
  if (mqtt.connected()) {
    return;
  }

  Serial.print("Connecting to MQTT... ");

  uint8_t retries = 3;
  while ((ret = mqtt.connect()) != 0) { // connect will return 0 for connected
       Serial.println(mqtt.connectErrorString(ret));
       Serial.println("Retrying MQTT connection in 5 seconds...");
       mqtt.disconnect();
       delay(5000);  // wait 5 seconds
       retries--;
       if (retries == 0) {
         // basically die and wait for WDT to reset me
         while (1);
       }
  }
  Serial.println("MQTT Connected!");
}

void sendRequest(String apt){

            //GET /prod/PostToSlack HTTP/1.1
            //HOST:   yf0u9kmss3.execute-api.us-west-1.amazonaws.com
            //https://yf0u9kmss3.execute-api.us-west-1.amazonaws.com/prod/PostToSlack

            if (!secureClient.connect(host, httpsPort)) {
                        Serial.println("connection failed");
                        return;
                      }
                    if (secureClient.verify(fingerprint, host)) {
                        Serial.println("certificate matches");
                      } else {
                        Serial.println("certificate doesn't match");
                      }

                    if(apt == "emergency"){
                                apt = "building";
                    }

              // Now we can publish stuff!
              Serial.println(F("Sending request"));
              secureClient.print(String("POST ") + url + " HTTP/1.1\r\n" +
            "Host: " + host + "\r\n" +
                                    "content-length: 10\r\n\r\n" +
                                    apt + "        \r\n"); //make sure spaces + apt.size <= 10


              Serial.println("request sent");

}


#endif /* PROJ_FUNCS_H_ */
```

Sasbot.cpp

```
#include "proj_globals.h"
#include "proj_funcs.h"
#include "proj_setup.h"

void loop() {

            MQTT_connect();
            //if not going to send data within 5 min, ping the server
//          if(!mqtt.ping()) {
//                      mqtt.disconnect();
//                    }

            //send request
```

```
                text = myNextion.listen();
                pageID = myNextion.pageId();
                if(text!=""){
                        Serial.println("");
                        Serial.println(text);
                        text.remove(0, 3); //remove leading 70 code

                         if(pageID == SENDINGPAGE || pageID == 0){
                                sendRequest(text);
                         }else if(pageID == PINPAGE){
                                Serial.println("Checking PIN");
                                if(text == PIN){
                                        myNextion.sendCommand(GRANT);
                                }else{
                                        myNextion.sendCommand(DENY);
                                        if(text == "1 9 9 3 "){
                                                wifiManager.resetSettings();
                                        }
                                }

                         }
                }


        //check for access
         while ((subscription = mqtt.readSubscription(500))) {
                 //check for access
                 if (subscription == &accessFeed) {
                access = (char*)accessFeed.lastread;
                if(access== "121"){
                        myNextion.sendCommand(GRANT);
                }else{
                        myNextion.sendCommand(DENY);
                }
                //check for update messages
          }else if(subscription == &postupdateApt1Feed){
                 Serial.print(F("Got from PostUpdate for Apt1: "));
                 updateMess = (char*)postupdateApt1Feed.lastread;
                 Serial.println(updateMess);
                 myNextion.setComponentText("kaywanPage.t0", updateMess);
          }else if(subscription == &postupdateApt2Feed){
                Serial.print(F("Got from PostUpdate for Apt2: "));
                updateMess = (char*)postupdateApt2Feed.lastread;
                Serial.println(updateMess);
                myNextion.setComponentText("ashotPage.t0", updateMess);
          }else if (subscription == &postupdateManagerFeed){
                Serial.print(F("Got from PostUpdate for Manager: "));
                updateMess = (char*)postupdateManagerFeed.lastread;
                Serial.println(updateMess);
                myNextion.setComponentText("yinPage.t0", updateMess);
                 }
          }


}
```

Code for AWS Lambda

(3 Lambda functions in Python)

PostToSlack

```
'''
Created on Mar 4, 2017

@author: Ash
'''

from __future__ import print_function
from slackclient import SlackClient
import time
import json
from CommonStrings import *


# Create an instance of the REST client.
ashapptoken = "Your App Token, for private channels. Owner of the token has to be in the group "
kayapptoken = "Your App Token, for private channels. Owner of the token has to be in the group"
bottoken = "Your Bot Token"
botsc = SlackClient(bottoken)
ashappsc = SlackClient(ashapptoken)
kayappsc = SlackClient(kayapptoken)
appsc = ashappsc

def lambda_handler(event, context):

    anounce = "Hey, Someone is at your door."
    apt = event['body']
    apt = apt.replace(" ", "")
    print(apt)
    if apt == "building":
        anounce = "Emergency service is at your door!"
    postStatus = json.dumps(botsc.api_call("chat.postMessage", username='botbot', icon_emoji=':robot_face:',
                    channel=apt, text=anounce, attachments=atta))
    postStatus = json.loads(postStatus)
    print(postStatus)
    print(postStatus['channel'])

    #sleep then check if the user responded
    if (apt == "apt1"):
        appsc = kayappsc
    else:
        appsc = ashappsc
    time.sleep(15)
    currentStatus = json.dumps(appsc.api_call("groups.history", channel=postStatus['channel'],
                                latest=postStatus['ts'],
                                oldest=postStatus['ts'], inclusive="true"))
    currentStatus = json.loads(currentStatus)
    print(currentStatus)
    if(not(currentStatus['messages'][0]['text']=="Access Granted" or
      currentStatus['messages'][0]['text']=="Access Denied")):
        #if no response mark as "expired"
        json.dumps(botsc.api_call("chat.update", username='botbot',
            ts=postStatus['ts'], channel=postStatus['channel'], text="Expired",
            attachments=None))
        #delete the message
        time.sleep(5)
        json.dumps(botsc.api_call("chat.delete",
            ts=postStatus['ts'], channel=postStatus['channel']))

    return {
        "statusCode": "200",
        "body": "Request Sent!",
        "headers": {
            "Content-Type": "application/json",
        },
    }
```

SlackButton

```
'''
Created on Mar 4, 2017

@author: Ash
'''

from __future__ import print_function
from Adafruit_IO import Client
import json
import urllib


# Set to your Adafruit IO key.
ADAFRUIT_IO_KEY = 'Your Adafruit IoT Key'
aio = Client(ADAFRUIT_IO_KEY)


def lambda_handler(event, context):

    payload = urllib.unquote(urllib.unquote(event['body']))

    payload = payload[8:]
    payload = json.loads(payload)
    print(payload)
    access = payload['actions'][0]['value']

    if(access == "grant"):
        update = 'Access Granted'
        aio.send('access', 121)
    else:
        update = 'Access Denied'
        aio.send('access', 120)
    return {
        'statusCode': '200',
        'body': update,
        'headers': {
            'Content-Type': 'application/json',
        },
    }
```

PostUpdate

```
'''
Created on Mar 4, 2017

@author: Ash
'''

from __future__ import print_function
from Adafruit_IO import Client
import json
import urllib


# Set to your Adafruit IO key.
ADAFRUIT_IO_KEY = 'Your Adafruit IoT Key'
aio = Client(ADAFRUIT_IO_KEY)


def lambda_handler(event, context):

    payload = urllib.unquote(urllib.unquote(event['body']))
    print(payload);
    message = payload[payload.find("text")+5:]
    message = message[:message.find("&")]
    message = message.replace("+", " ")
    print(message)
    channel = payload[payload.find("channel_id")+11:]
    channel = channel[:channel.find("&")]
    print(channel)
    if channel == "Your Channel":
        channel = "manager"
    elif channel == " Your Channel":
        channel = "apt1"
    elif channel == " Your Channel":
        channel = "apt2"
    else:
        return {
            'statusCode': '200',
            'body': 'Cannot post updates here.',
            'headers': {
                'Content-Type': 'application/json',
            },
        }
```

```
channel = "postupdate" + channel
aio.send(channel, message)

return {
    'statusCode': '200',
    'body': 'Updated!',
    'headers': {
        'Content-Type': 'application/json',
    },
}
```

References

Project Site	https://github.com/Ashoth
3D Enclosure	https://www.thingiverse.com/thing:1527472
Eclipse Plug-in	https://marketplace.eclipse.org/content/eclipse-c-ide-arduino
Nextion Display	http://imall.itead.cc/nextion-nx2432t043.html
Nextion Library	https://github.com/bborncr/nextion
Slack API	https://api.slack.com
AWS Lambda	https://aws.amazon.com/lambda